# NAVAL POSTGRADUATE SCHOOL

## Monterey, California


# THESIS



DATA COMPRESSION AND ARCHIVING SOFTWARE IMPLEMENTATION AND THEIR ALGORITHM COMPARISON

by

Young Je Jung

March, 1992

Thesis Advisor: Chyan Yang

Approved for public release; distribution is unlimited

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution is unlimited. |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION<br>Naval Postgraduate School<br>6b OFFICE SYMBOL (*If applicable*)<br>32 | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| 6c ADDRESS (*City, State, and ZIP Code*)<br>Monterey, CA 93943-5000 | 7b ADDRESS (*City, State, and ZIP Code*)<br>Monterey, CA 93943-5000 |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION<br>8b OFFICE SYMBOL (*If applicable*) | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c ADDRESS (*City, State, and ZIP Code*) | 10 SOURCE OF FUNDING NUMBERS<br>Program Element No / Project No / Task No / Work Unit Accession Number |

11 TITLE (*Include Security Classification*)
DATA COMPRESSION AND ARCHIVING SOFTWARE IMPLEMENTATION AND THEIR ALGORITHM COMPARISON

12 PERSONAL AUTHOR(S) Young Je Jung

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (*year, month, day*) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | From To | March 1992 | 94 |

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 17 COSATI CODES | | | 18 SUBJECT TERMS (*continue on reverse if necessary and identify by block number*) |
|---|---|---|---|
| FIELD | GROUP | SUBGROUP | Data Compression and Archiving, Software Performance Analysis, Compression Ratios and Execution Time Comparison, Algorithm Comparison |
| | | | |
| | | | |

19 ABSTRACT (*continue on reverse if necessary and identify by block number*)

Although data compression has been studied for over 30 years, many new techniques are still evolving. There is considerable software available that incorporates compression schemes and archiving techniques. The U.S. Navy is interested in knowing the performance of this software. This thesis studies and compares the software. The testing files consist of the file type specified by the U.S. Naval Security Detachment at Pensacola, Florida.

| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21 ABSTRACT SECURITY CLASSIFICATION | |
|---|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS | Unclassified | |
| 22a NAME OF RESPONSIBLE INDIVIDUAL<br>Chyan Yang | 22b TELEPHONE (*Include Area code*)<br>(408)646-2266 | 22c OFFICE SYMBOL<br>Code EC/Cw |

**DD FORM 1473, 84 MAR** 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

# Data Compression and Archiving Software Implementation and Their Algorithm Comparison

by

Young Je Jung
Captain, Korean Army
B.S., KumOh Institute of Technology

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

## ABSTRACT

Although data compression has been studied for over 30 years, many new techniques are still evolving. There is considerable software available that incorporates compression schemes and archiving techniques. The U.S. Navy is interested in knowing the performance of this software. This thesis studies and compares the software. The testing files consist of the file types specified by the U.S. Naval Security Detachment at Pensacola, Florida.

C.1

## TABLE OF CONTENTS

**ACKNOWLEDGEMENT**

I'd like to express my appreciation to the Army of the Republic of Korea which has given me a great opportunity for graduate studies and physical support.

I wish to thank my thesis advisor, Professor Chyan Yang, for his sincere help and effort to improve my thesis, and to my second reader, Professor Glen Myers, for his encouragement to study hard.

I also appreciate Lt.Col. Tsai who has given me a lot of advice and help. I should not omit Hal Lasell, who helped me obtain software packages and experimental files.

Finally, I give thanks to my parents, my wife's parents, my lovely wife, Sung Mi, and my heart, Eunjoo(Grace) who was born during my graduate studies.

Thank God.

# I. INTRODUCTION

In file management, one may use data compression and archiving for cost reduction in data storage and transmission. In other words, the collection and analysis of data can reap benefits from compression. There are numerous kinds of data compression and archiving schemes. Popular software for data compression are StacPack, ARC, BTLZ, PKZIP, Splay, SHRINK, DIET, PKLTE, ARJ, LHA, PAK, ZOO, PKPAK, and LZEXE [Ref.3,6,7,8,12,13,14,15,16,17,22,23]. Some of these are solely for executable files while others are good for binary graphic files. Additionally, each software may have its own set of operating environment and performance edge. No two are identical. The Naval Security Group Detachment in Pensacola, Florida expressed its interest in evaluating public available compression software [Ref.24]. It is therefore interesting and desirable to compare the performance of each software in the Naval operating environment.

In this thesis, 3 methods for compression, and 4 methods for compression with archiving are chosen for comparing. The PKZIP package is examined for both compression and compression with archiving. This thesis focuses on reversible data compression: the original file can be completely recovered from the compressed file.

The benefits of data compression are many. First, hardware costs can be cut back because of the reduced capacity requirement for disk drive units. Second, given a fixed amount of disk space, more data can be kept online. Third, the speed of effective data transfer can be increased while reducing costs when copying files to disks or tapes, sending data over communications equipment, and shipping data recorded on disks or tapes. Fourth, the amount of media (e.g. tapes) to archive the data offline can be reduced. Last, as a result of the compression process, compressed files are encrypted; therefore, they automatically acquire greater protection from unauthorized access [Ref.13]. The trade-off for the benefits is mainly in execution time. The more effective compression algorithms generally need more CPU overhead than the less effective ones [Ref.13]. The result of experiments conducted in this thesis shows that a good archiving program generally results in good performance in data compression.

This thesis is organized as follows. Chapter II discusses the generic compression algorithms while Chapter III examines the algorithms used in each software package. The main effort of data compilation and analysis are presented in Chapter IV. Concluding remarks can be found in Chapter V.

## II. GENERIC COMPRESSION ALGORITHMS

In this chapter, several algorithms for data compression are introduced. These algorithms are already employed in commercial software. The compression ratios and archiving effectiveness of these commercial software packages will be compared and analyzed in Chapter IV.

### A. INTRODUCTION TO DATA COMPRESSION

Data compression is often referred to as source coding. Information theory is defined as the study of efficient coding and its consequences in the form of speed of transmission and probability of error. Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted [Ref. 10].

With most file types, some recurring patterns of bytes or words (redundancy) can be found. This effect can be optimized in a compressed file with symbols which indicate to the decompression program the particular pattern to restore at that location. The simplest and most common pattern, regardless of file type, is a string of repeating single characters or binary words. Most often these are strings of blanks which occur between words, statements, and paragraphs in text files. Other forms of redundancy tend to be more file-type specific. COBOL source code, for example, is

partially composed with a known set of reserved words which occur with great frequency within each program.

Once all the redundancies have been detected, the encoding algorithms, static or dynamic, can be used to code these redundancies. There always remains a core of information which cannot be compressed further. A compressed file contains the information which distinguishes it from any other file. At this point, the file can not be further reduced without some loss of information.

Most compression algorithms use a start-to-finish operation, that is, the entire file must be processed as a single unit. The entire file must be decompressed in order to access it. This scheme renders the use of data compression with production files inconvenient. An additional drawback to compressing information might be that compressed files are more susceptible to corruption. Particularly with start-to-finish algorithms, decompression requires a precise sequence of operations, which is exactly the reverse of the compression sequence. If this sequence is disrupted by a few corrupted bits on the storage media, it is possible to lose the remainder of the file. However, the reliability of current storage hardware makes this risk rather small [Ref.13].

No single technique described in the following section is the best in all situations. Typically, a sophisticated compression product will combine several of the following

methods as well as other techniques in the effort to extract every last unnecessary bit out of a compressed file.

## B. STATIC HUFFMAN CODING

The main idea behind Huffman coding is based on the frequency of occurrence of a symbol in the text. Symbol is defined as a particular sequence of bits. The most frequently used symbols are assigned a shorter binary pattern and less frequently symbols are assigned a longer pattern.

A static method is one in which the mapping from the set of codewords is fixed before transmission begins so that a given message is represented by the same codeword every time it appears in the message ensemble [Ref.10].

Huffman's algorithm, expressed graphically, takes as input a list of nonnegative weights $\{w_1, \ldots, w_n\}$ and constructs a full binary tree - a binary tree is full if every node has either zero or two branches - whose leaves are labeled with the weights. When the Huffman algorithm is used to construct a code, the weights represent the probabilities associated with the source letters. Initially, there is a set of singleton trees, one for each weight in the list. At each step in the algorithm the trees corresponding to the two smallest weights, $w_i$ and $w_j$, are merged into a new tree whose weight is $w_i + w_j$ and whose root has two branches that are the subtrees represented by $w_i$ and $w_j$. The weights $w_i$ and $w_j$ are removed from the list, and $w_i + w_j$ is inserted into the list. This

process continues until the weight list contains a single value. If, at any time, there is more than one way to choose a smallest pair of weights, any such pair may be chosen. In Huffman's paper the process begins with a nonincreasing list of weights. This detail is not important to the correctness of the algorithm, but it does provide a more efficient implementation. The Huffman algorithm is demonstrated in Figure 1 and Figure 2 [Ref.10].

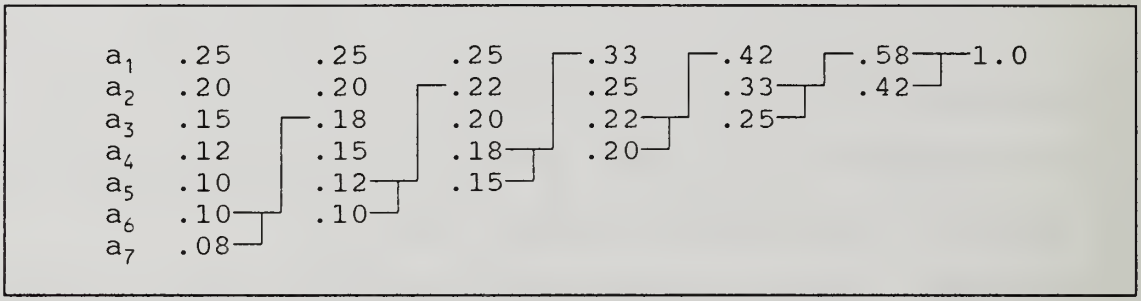


**Fig. 1.** The List of Huffman Process.

The Huffman algorithm determines the lengths of the codeword to be mapped to each of the source letters $a_i$. There are many ways for specifying the actual bits; it is necessary only that the code have the prefix property. The usual assignment entails labeling the edge from each tree to its left branch with the bit *0* and the edge to the right branch with *1*. The codewords for each source letter are the sequence of labels along the path from the root to the leaf node representing that letter. The codewords that can be generated from Figure 2, in order of decreasing probability, are *{01, 11, 001, 100, 101, 0000, 0001}*. Clearly, this process yields

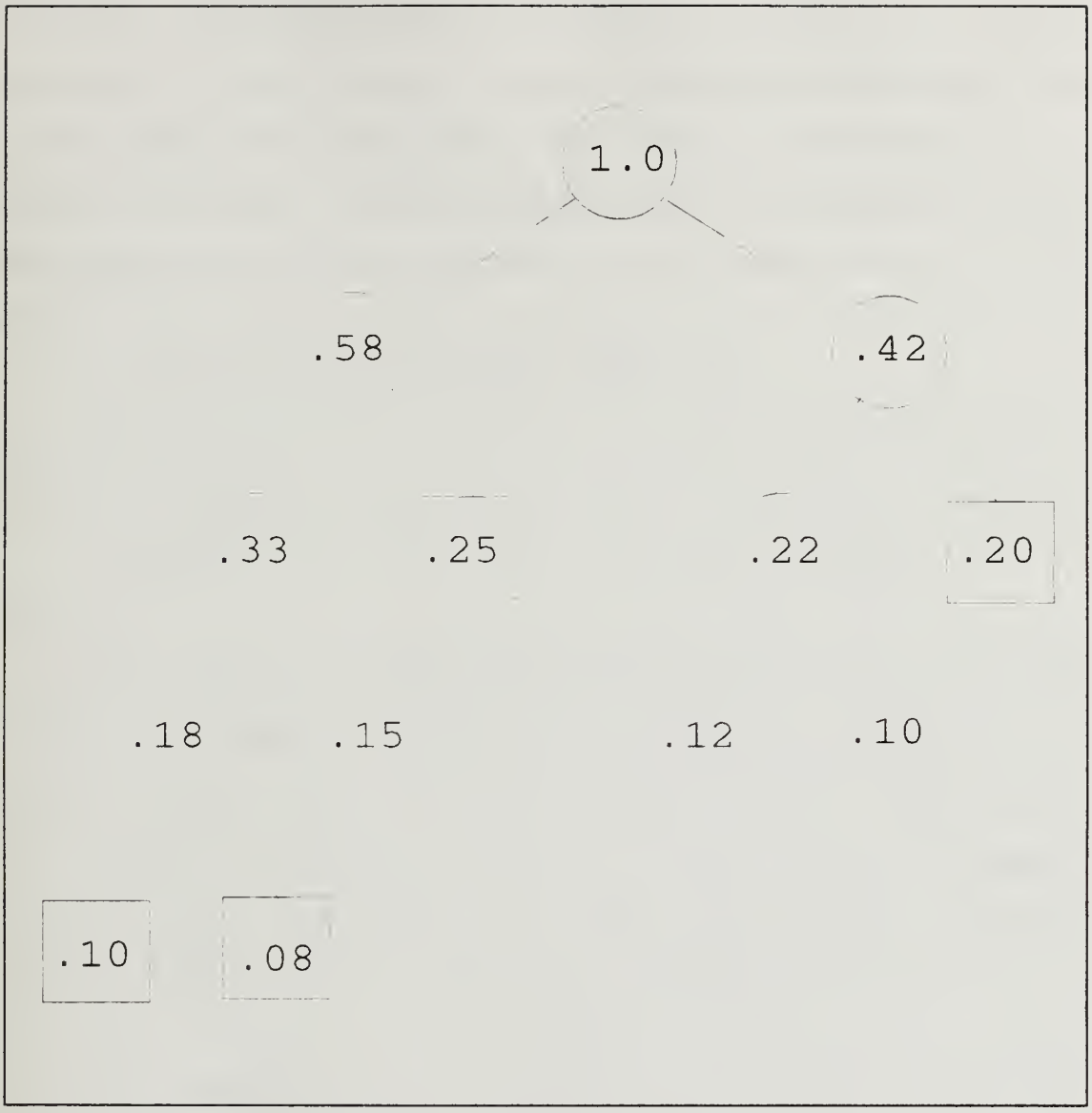


**Fig. 2.** The Tree of The Huffman Process.

a minimal prefix code. Furthermore, the algorithm is guaranteed to produce an optimal (minimum redundancy) code. Gallager has proved an upper bound on the redundancy of a Huffman code equal $P_n + \log[(2 \log e)/e] \approx P_n + 0.086$, where $P_n$ is the probability of the least likely source message

[Ref.10]. Figure 3 shows the distribution for which the Huffman code is optimal.

In addition to the fact that there are many ways of forming codewords of appropriate lengths, there are cases in which the Huffman algorithm does not uniquely determine these lengths owing to the arbitrary choice among equal minimum weights. For example, codes with codeword lengths of *{1,2,3,4,4}* and *{2,2,2,3,3}* both yield the same average codeword length for a source with probabilities *{.4, .2, .2, .1, .1}*. Schwartz defines a variation of the Huffman algorithm that performs "bottom merging", that is, that orders a new parent node above existing nodes of the same weight and always merges the last two weights in the list. The code constructed is the Huffman code with minimum values of maximum codeword length ($\max\{l_i\}$) and total codeword length ($\Sigma l_i$). Schwartz and Kallick describe an implementation of Huffman's

| | | |
|---|---|---|
| $a_1$ | 0.35 | 1 |
| $a_2$ | 0.17 | 011 |
| $a_3$ | 0.17 | 010 |
| $a_4$ | 0.16 | 001 |
| $a_5$ | 0.15 | 000 |
| Average codeword length | | 2.30 |

**Fig. 3.** Distribution of Huffman Code.

algorithm with bottom merging. The Schwartz-Kallick algorithm and a later algorithm by Connell use Huffman's procedure to determine the lengths of the codewords, and actual digits are assigned so that the code has the numerical sequence property;

that is , codewords of equal length form a consecutive sequence of binary numbers. Shannon-Fano codes also have the numerical sequence property. This property can be exploited to achieve a compact representation of the code and rapid encoding and decoding [Ref.10].

## C. LZ77 OPM/L TEXT COMPRESSION TECHNIQUE

Lempel-Ziv coding represents a departure from the classic view of a code as a mapping from a fixed set of source messages(letters, symbols, or words) to a fixed set of code-words.

One of the popular data-compression algorithms, suggested by Ziv and Lempel is the OPM/L (Original Pointer Macro restricted to Left Pointers), LZ77 [Ref.2]. OPM/L uses sliding-window dictionary (SWD), a variation of the Lempel-Ziv-Welch (LZW) algorithm. The basic idea behind SWD is simple: substrings of the input stream are stored in a dictionary. Each dictionary entry is assigned a value. Then, if a later section of the input stream is found within the dictionary, the value of this dictionary entry is substituted in place of the longer original data.

The OPM/L scheme replaces a substring in a text with a pointer to a previous (left) occurrence of the substring in the text. The pointer represents the position and size of the substring in the original text. These restrictions make fast single-pass decoding straightforward [Ref.2].

The LZ77 scheme restricts the reach of the pointer to approximately the previous *N* characters, effectively creating a "window" of *N* characters which is used as a sliding dictionary. Pointers are chosen using a "greedy" algorithm which permits single-pass encoding [Ref.2]. Following are advantages of using window:

1) The amount of memory required for encoding and decoding is bounded by the size of the window, and is typically no more than *8* kbytes;
2) For many types of text, and for sufficiently large *N*, the window is a good dictionary for the substring which follows, because it will usually contain the same language, style, and topic; and
3) All pointers can have fixed size fields.

An LZ77 encoder is parameterized by *N*, the size of the "window", and *F*, the maximum length of a substring that may be replaced by a pointer. Encoding of the input string proceeds from left to right. At each step of the encoding, a section of the input text is available in a window of *N* characters. Of these, the first *N-F* characters have already been encoded and the last *F* characters are the "lookahead buffer" [Ref.2].

For example [Ref.2], if the string *s* = *abcabcabcabcabcabc...*

is being encoded with the parameters *N = 11* and *F = 4* and character *12* is to be encoded next, the window is shown as Figure 4.

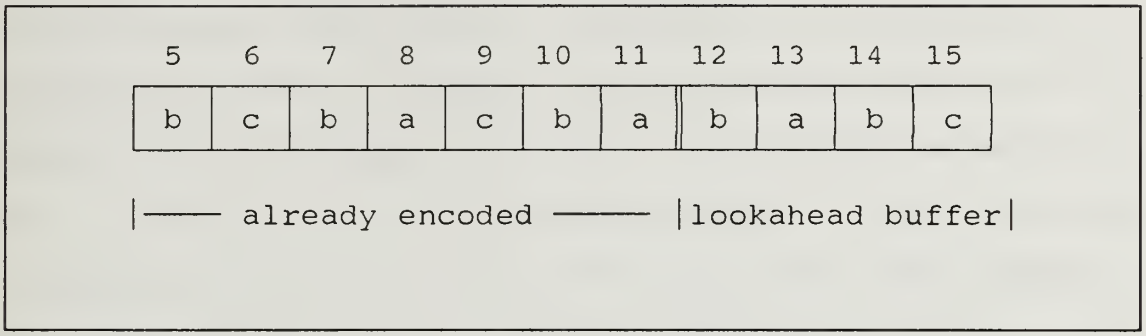


**Fig. 4.** LZ77 Encoding String Window.

Initially the first *N - F* characters of the window are (arbitrary) blanks, and the first *F* characters of the text are loaded into the lookahead buffer.

The already encoded part of the window is searched to find the longest match for the lookahead buffer. The match may overlap with the lookahead buffer, but obviously cannot be the lookahead buffer itself. In the example, the longest match for the "babc" is "bab", which starts at character *10*.

The longest match is then coded into a triple *<i,j,a>*, where *i* is the offset of the longest match from the lookahead buffer, *j* is the length of the match, and *a* is the first character which did not match the substring in the window. In the example, the output triple would be *<2,3,'c'>*. The window is then shifted right *j + 1* characters, ready for another coding step.

A window of moderate size, typically $N \leq 8192$, can work well for a variety of texts for the following reasons:

1) Common words and fragments of words occur regularly enough in a text to appear more than once in a window. For example, in English "the," "of," "pre-," "-ing,"; source program keywords "while," "if," "then."
2) Specialist words tend to occur in clusters. For example, a paragraph on a technical topic, or local identifiers in a procedure of a source program.
3) Less common words may be made up of fragments of common words.
4) Runs of characters are coded compactly. For example, *k* blanks may be coded recursively as *<?, ?, ' '> <1, k-1, ?>*. The amount of memory required for encoding and decoding is limited to the size of the window. The offset *(i)* in a triple can be represented in $[\log_2 (N-F)]$ bits, and the number of characters *(j)* covered by the triple in $[\log_2 F]$ bits. The time taken at each step is bounded to $N - F$ substring comparisons, which is constant, so the time used for encoding is $O(n)$ for a text of size $n$ [Ref.2].

Decoding is very simple and fast. The decoder maintains a window in the same way as the encoder but, instead of searching for a match in the window, it copies the match from the window using the triple given by the encoder [Ref 2].

The main disadvantage of LZ77 is that, although the encoding step requires *O(1)* time, a straightforward implementation can require up to *(N - F)*F* character comparisons, typically on the order of several thousands. LZ77 is therefore best for the situation where a file is to be encoded once (preferably on a fast computer) and decoded many times, possibly on a small machine [Ref.2].

LZSS, a slightly modified version of LZ77 which improves the compression ratios for a wide range of text was developed by Storer and Szymanski. It offers very fast decoding but requires comparatively little memory for coding and decoding [Ref.18].

Storer and Szymanski presented a general mode for data compression that encompasses Lempel-Ziv coding. Their broad theoretical work compares classes of 'macro schemes', where macro schemes include all methods that factor out duplicate occurrences of data and replace them by references either to the source ensemble or to a code table. They also contribute a linear-time Lempel-Ziv-like algorithm with better performance than the standard Lempel-Ziv method [Ref.10].

### D. ARITHMETIC CODING

At present, most of the commonly used data compression methods fall into one of two categories: dictionary-based schemes or statistical methods. In the world of small systems, dictionary-based data compression techniques seem to

be more popular. However, by combining arithmetic coding with powerful modeling techniques, statistical methods for data compression are actually able to achieve better performance [Ref.10].

The method of arithmetic coding was suggested by Elias and presented by Abramson [Ref.10] in his text on information theory. Implementations of Elias' technique were developed by Risssanen, Pasco, Rubin, and, most recently, Written et al.

Arithmetic coding is based on the idea that each symbol is not coded independently one after another as in a Huffman code, but coded as a portion of the real interval between *0* and *1*. Each symbol of the ensemble narrows this interval. As the interval becomes smaller, the number of bits needed to specify it grows. Arithmetic coding assumes an explicit probabilistic model of the source. It is a defined-word scheme that uses the probabilities of the source messages to successively narrow the interval used to represent the ensemble. A high-probability message narrows the interval less(faster) than a low-probability messages, and contributes fewer bits to the coded message. The method begins with an unordered list of source messages and their probabilities. The number line is partitioned into subintervals on the basis of cumulative probabilities.

It is instructive to see an example [Ref.10]. Given source messages *{A,B,C,D,#}* with probabilities *{.2, .4, .1, .2, .1}*, Table I shows the initial partitioning of the number line [0,

1]. The symbol *A* corresponds to the first *1/5* of the interval *[0,1)*, *B* is the next *2/5*, and *D* is the subinterval of size *1/5* which begins at *70%* of the interval from the left endpoint.

**Table I** The arithmetic coding model

| Source message | Prob. | Cumul.Prob. | Range |
|---|---|---|---|
| A | .2 | .2 | [0,.2) |
| B | .4 | .6 | [.2,.6) |
| C | .1 | .7 | [.6,.7) |
| D | .2 | .9 | [.7,.9) |
| # | .1 | 1.0 | [.9,1.0) |

When encoding begins, the source ensemble is represented by the entire interval *[0,1)*. For the ensemble *AADB#*, the first *A* reduces the interval to *[0,.2)* and the second *A* to *[0,.04)* (the first *1/5* of the previous interval or *0.2×0.2[0, .2]*). *D* further narrows the interval to *[.028, .036)*(*1/5* of the previous size, beginning *70%* of the distance from left to right or *0.2×0.2×[0.7, 0.9]*). *B* narrows the interval to *[.0296, .0328)*(*2/5* of the previous size, *[.028, .036]*, beginning *20%* and ending *60%* of the distance from left to right, *[.028+.0016, .028+.0048]*) and the # yields a final interval of *[.03248, .0328)*. The interval, or alternatively any number *i* within the interval, may now be used to represent the source ensemble.

Two equations may be used to define the narrowing process described above:

$$newleft = prevleft + msgleft \times prevsize \quad (1)$$

$$newsize = prevsize \times msgsize \quad (2)$$

Equation (1) states that the left endpoint of the new interval is calculated from the previous interval and the current source message. The left endpoint of the range associated with the current message specifies what percent of the previous interval to remove from the left in order to form the new interval. For character *D* in the above example (*AADB#*), the new left endpoint is moved by $.7 \times .04$ (*70%* of the size of the previous interval). Equation (2) computes the size of the new interval from the previous interval size and the probability of the current message (which is equivalent to the size of its associated range). Thus, the size of the interval determined by *D* is $.04 \times .2$, and the right endpoint is $.028 + .008 = .036$ (left endpoint + size).

The size of the final subinterval determines the number of bits needed to specify a number in that range. The number of bits needed to specify a subinterval of *[0, 1)* of size *s* is:

$$k = -\log_2 s$$

Since the size of the final subinterval is the product of the probabilities of the source messages in the ensemble:

$$s = \prod_{i=1}^{N} P(\textit{source message } i)$$

$N$ : length of the ensemble

we have:

$$- \log_2 s = - \sum_{i=1}^{N} \log_2 P(\textit{source message } i)$$

$$= - \sum_{i=1}^{N} P(a_i) \log_2 P(a_i)$$

$n$ : number of unique source messages $a_1$, $a_2$, ..., $a_n$

Thus, the number of bits generated by the arithmetic coding technique is exactly equal to the entropy. This demonstrates the fact that arithmetic coding achieves compression which is almost exactly that predicted by the entropy of the source.

In order to recover the original ensemble, the decoder must know the mode of the source used by the encoder (e.g., the source messages and associated ranges) and a single number within the interval determined by the encoder. Decoding consists of a series of comparisons of the number *i* to the ranges representing the source messages. For the example of *AADB#*, *i* might be *.0325* or a number in *[.03248, .0328]*. The decoder uses *i* to simulate the actions of the encoder. Since *i* lies between *0* and *.2*, the decoder deduces that the first letter was *A* (since the range is *[0,.2]*). The decoder can now deduce that the next message will further narrow the interval in one of the following ways: to *[0,.04)* for *C*, to *[.14,.18)* for *D*, or to *[0,.04)*; the decoder knows that the second

message is again *A*. This process continues until the entire ensemble has been recovered [Ref.10].

## E. SHANNON-FANO CODING

As one of the optimum source coding scheme with Huffman code, Shannon-Fano code is known for its reasonable efficiency with instantaneous decodability. Shannon-Fano coding is a variable length coding process. Before one decides the code for each character, one has to determine the probability of the occurrence of each character and then arrange the source message in descending order, which is based on the probability of occurrence of each character. Once it is done, the character set(source message) must be divided into two subsets of equal, or almost equal, probability. The first

**Table II** Shannon-Fano Coding

| Charac. | Prob. | Descending | | Prob. | Code | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | 0.10 | $C_7$ | → | 0.25 | 1 | 1 | | | |
| $C_2$ | 0.05 | $C_3$ | → | 0.20 | 1 | 0 | | | step 1 |
| $C_3$ | 0.20 | $C_6$ | → | 0.15 | 0 | 1 | 1 | | |
| $C_4$ | 0.10 | $C_1$ | → | 0.10 | 0 | 1 | 0 | | step 2 |
| $C_5$ | 0.05 | $C_4$ | → | 0.10 | 0 | 0 | 1 | | step 3 |
| $C_6$ | 0.15 | $C_8$ | → | 0.10 | 0 | 0 | 0 | 1 | step 4 |
| $C_7$ | 0.25 | $C_2$ | → | 0.05 | 0 | 0 | 0 | 0 | 1 |
| $C_8$ | 0.10 | $C_5$ | → | 0.05 | 0 | 0 | 0 | 0 | 0 |

digit in one subset is assigned a binary *0* value while a binary *1* is assigned as the first digit in the second subset. This process of forming subsets is continued until the character set is completely subdivided. Finally, a suffix bit is added to each character in a two-character subset as required to distinguish one character's binary composition from the other character in the subset [Ref.10].

To help understand Shannon-Fano coding, consider the following example [Ref.9: p.107-109]. It is assumed the character set contains *8* characters with the probabilities given in Table II.

The third column of Table II is the character set arranged in descending order based upon the probabilities. To form the

**Table III** An Example of a Completed Shannon-Fano Code

| Character | Probability | Code |
|---|---|---|
| $C_7$ | 0.25 | 1 1 |
| $C_3$ | 0.20 | 1 0 |
| $C_6$ | 0.15 | 0 1 1 |
| $C_1$ | 0.10 | 0 1 0 |
| $C_4$ | 0.10 | 0 0 1 |
| $C_8$ | 0.10 | 0 0 0 1 |
| $C_2$ | 0.05 | 0 0 0 0 1 |
| $C_5$ | 0.05 | 0 0 0 0 0 |

subsets, we have to group the characters in them so that they are equal or as nearly equal as possible. We next assign

binary *1*'s to one subset and binary *0*'s to the other subset and continue the process until all possible subsets are constructed. The fifth column of Table II shows the process [Ref.9].

**F. LZW CODING**

This is one of the modified version of Lempel-Ziv, which involves the way in which the string table is stored and accessed [Ref.10].

Welch described the implementation of this algorithm known as the LZW algorithm. It has the advantage of being adaptive. That is, the algorithm does not assume any advance knowledge of the properties of the input and builds the dictionary used for compression only on the basis of the input as it is read. This property is especially important in compression for communication. This method contrasts compression algorithms which are based on advance knowledge of the properties of the input, e.g. Huffman algorithm [Ref.19].

The LZW algorithm starts with a dictionary containing entries for each character in the alphabet. The algorithm scans the input matching it with entries in the dictionary. The matching is finished, such that $Y = X.a$, where $X$ is a string already in the dictionary, "a" is a character and "." denotes the concatenation operation. The compression algorithm then sends the code for $X$ (an index into the dictionary table) and inserts $Y$ into the dictionary. The string $Y$ is called a

character extension of $X$. The encoding of the input continues from the character "a" that follows $X$. Meanwhile, the decoder builds an identical dictionary to the one built by the encoder [Ref.19].

The entries for the LZW dictionary satisfy the two properties: 1) If a string $X$ is in the dictionary then every prefix of $X$ is also in the dictionary. 2) For every code sent by the encoder, a new entry is added to the dictionary. Since the dictionary size is finite and may be limited for practical reasons, the dictionary may fill up fast. The LZW algorithm then continues by encoding according to the existing dictionary without adding new entries to it. Experiments show that after a certain time, a significant decline in the compression ratio may be observed. This decline is typically due to a change in the properties of the text so that the dictionary is no longer appropriate. At this point the LZW algorithm forgets the old dictionary and starts from scratch, usually obtaining again a higher compression ratio [Ref.19].

It is helpful to look at the representation of the dictionary as an ordered labeled rooted tree. Each edge emanating from a vertex is labeled by a character of the alphabet. A vertex represents the string obtained by concatenation of all the characters along the path from the root to the vertex. Thus all vertices on the path from the root to a vertex representing a string $X$ of the dictionary represent prefixes of $X$ and their corresponding strings are

also in the dictionary. Using this tree representation, if the string of a vertex is deleted then the strings of all its descendants must also be deleted. Note that when the dictionary is full, the degree of a vertex is equal to the number of times the corresponding entry was sent. Hence a leaf represents an entry which was inserted into the dictionary but was never sent. Depending on the nature of the text and size of the dictionary, a commercial program called COMPRESS written in 'C' language and based on the LZW algorithm yields compression ratios of up to *60%*. The "compression ratio" is defined as the difference between the number of characters in the original text and the compressed text divided by the number of characters in the original text.

The dictionary constructed by the LZW algorithm contains variable length strings of consecutive characters from the text. Compression is obtained due to the replacement of the text strings by the index to the corresponding dictionary entry. For example if the dictionary size is $2^{10}$, it can encode any string in the dictionary using just *10* bits [Ref. 19].

## III. COMMERCIAL OR PUBLIC ALGORITHMS

### A. AN OVERVIEW OF COMPRESSION SOFTWARE

As MS-DOS became the dominant operating system of personal computers, data storage capacities also increased. Hard disk drives with capacities of over *40* Mbytes became commonly available. Additionally, the *1200-Kbit/second* modems are now available for less than *$1000*. Despite these advances in data storage and data communications, the sheer volume of data files continues to outpace the new technology's ability to provide adequate storage.

With MS-DOS, the necessity for new data compression softwares become evident. The first important application was System Enhancement Associates' (SEA) ARC, which for many years was the popular program for data compression. Like many other DOS compression programs, ARC was shareware: software distributed through the online community without charge [Ref.12].

Continually, better programs have been introduced - notably PKware's PKARC and PKZIP - and SEA's ARC lost its dominance in the field [Ref.12].

Today there are at least half a dozen MS-DOS archival/compression programs. PKZIP 1.10 may be the fastest and most efficient of these programs, though NoGate

Consulting's PAK 2.6 also offers outstanding performance. LHARC 1.13C, a popular compression program originated in Japan,[Ref.12] is almost as good as PKZIP except it runs slower than PKZIP.

Another notable program is ZOO 2.01 [Ref.12]. Using a Lempel-Ziv compression algorithm, it was developed by R. Dhesi [Ref.23]. ZOO 2.01 neither runs fast nor compresses as well as other programs; its compression ratio for text files is about *10%* less than that of PKZIP. However, it has some unique advantages. Originated in Unix, it has since been ported to nearly every operating environment [Ref.12].

There are still many problems related to data compression that remain to be solved. For example, error detection and error correction are not incorporated in most software packages.

Every time one compresses a file using a package, the package will confirm whether the compressed file has lost some of its data or not. Both compressed and uncompressed files can fail because a disk has marginal sectors or because of some "accident". If the file contains executable code, there's no point in fixing it - one can simply restore it from a backup. But if the file contains data, it is often possible and worthwhile to recover the rest, even though a few bits or a sector may be missing. When a compressed file goes bad, recovery is harder. Since the file is compressed, the damage is multiplied. Naturally, the compression program should have

a decompress function; otherwise, there's no way one can recover the file back to the original format.

**Table IV** Comparison of Software and the Algorithm Employed

| Software | Algorithm | Remarks |
|---|---|---|
| PKZIP | LZW | Shrink |
| | LZ77 (SW) | Reduce(v0.9) |
| | LZ77 / Shannon-Fano | Implode |
| StacPack | | QIC |
| Compress | LZW | |
| ARJ221A | LZ77 (SW) | |
| LHA213 | (S)Huffman | |
| PAK251 | Huffman/LZ77 | Distill |
| | LZW | Crush |

Table IV summarizes the algorithms used by each software package. The algorithm used by StacPack was not disclosed by the company.

## B. GENERAL DESCRIPTION OF EACH SOFTWARE

### 1. PKZIP

This is one of the commercial compression techniques that is widely used and known. Version 1.1 composed by P. Katz, PKWARE Inc., uses a proprietary dictionary-based scheme.

One must have PKUNZIP to extract compressed and archived files. This version claims to be faster in compressing very large files and exhibits good compression efficiency.

*a. Compression Algorithm*

PKZIP has *3* different kind of compression techniques: Shrinking, Reducing, and Imploding. As mentioned in Table IV, they employ several algorithms such as LZW, LZ77, and Shannon-Fano coding.

**Shrinking** is a Dynamic Ziv-Lempel-Welch compression algorithm with partial clearing. The initial code size is *9* bits, and the maximum code size is *13* bits. Shrinking differs from conventional Dynamic Ziv-Lempel-Welch implementations in several aspects:

1) The code size is controlled by the compressor, and is not automatically increased when codes larger than the current code size are created (but not necessarily used). The decompressor should not increase the code size used until the sequence *256, 1* is encountered.
2) When the table becomes full, total clearing is not performed. Rather, when the compressor emits the code sequence *256,2*(decimal), the decompressor should clear all leaf nodes from the Ziv-Lempel tree, and continue to use the current code size. The nodes that are cleared from the Ziv-Lempel tree are then reused, with the lowest code value reused first, and the highest code value reused

last. The compressor can emit the sequence *256,2* at any time [Ref.8].

**Reducing** is a combination of two distinct algorithms. The first algorithm compresses repeated byte sequences, and the second algorithm takes the compressed stream from the first algorithm and applies a probabilistic compression method. The probabilistic compression stores an array of 'follower sets' *S(j)*, for *j=0* to *255*, corresponding to each possible ASCII character. Each set contains between *0* and *32* characters, to be denoted as *S(j)[0],...,S(j)[m]*, where *m<32*. The sets are stored at the beginning of the data area for a reduced file, in reverse order, with *S(255)* first, and *S(0)* last. The sets are encoded as *{ N(j), S(j)[0],...,S(j)[N(j)-1] }*, where *N(j)* is the size of set *S(j)*. *N(j)* can be *0*, in which case the follower set for *S(j)* is empty. Each *N(j)* value is encoded in *6* bits, followed by *N(j)* eight bit character values corresponding to *S(j)[0]* to *S(j)[N(j)-1]* respectively. If *N(j)* is *0*, then no values for *S(j)* are stored, and the value for *N(j-1)* immediately follows. Immediately after the follower sets is the compressed data stream. The compressed data stream can be interpreted for the probabilistic decompression [Ref.8].

**Imploding** is actually a combination of two distinct algorithms. The first algorithm compresses repeated byte sequences using a sliding dictionary. The second algorithm is

used to compress the encoding of the sliding dictionary output, using multiple Shannon-Fano trees [Ref.8].

*b. General Format of Zipped File*

When we look at the list of archived files, there are Length, Method, Size, Ratio, Date, Time, CRC-32, Attr, and Name. Those factors show the general format of PKZIP. The overall zipfile format is

*[local file header + file data]...*
*[central directory] end of central directory record*

Local file header is composed of *30* bytes of fixed factors including compression method, variable size of filename, and extra field. The structure of the central directory is *46* bytes of fixed factors including file comment length, variable size of file name, extra field, and file comment. End of central directory record consists of *22* bytes of fixed factors including end of central directory signature and variable size of zipfile comment.

The Length is the compressed size of each file. The compression method is dependent upon the characteristics of the data file. The file is stored only when it does not need compression or can not compress. The data and time are encoded in standard MS-DOS format. CRC-32 algorithm was contributed by David Schwaderer and can be found in his book "C Programmers

Guide to NetBios" published by Howard W. Sams & Co. Inc. For every file put in an archive, CRC (Cyclical Redundancy Check) is calculated and is recalculated when the file is extracted. It is done due to the necessity of ensuring data integrity when archives are transmitted over communication links. The lowest bit of internal file attributes confirms whether the data file is ASCII or binary. The size of the entire .ZIP file header, including the file name, comment, and extra filed would exceed *64K* in size [Ref.8].

**2. StacPack**

*a. PC Backup Program*

Stac Inc. provides 'Stacker' package for compressing disk files in real time. This company also provides data compression integrated circuit chips. The core of the 'Stacker' is a compression program StacPack and a decompression program **StacUnpk**. This program is also licensed to vendors that are in PC backup business. The backup routines in such popular DOS programs as Norton Backup and PC Tools are built on StacPack's algorithm [Ref.12].

*b. QIC - 122*

StacPack's algorithm has proven to be so successful that the Quarter-Inch Cartridge (QIC) Consortium has adopted it as a standard, known as *QIC-122*, for QIC tape drives. With StacPack, tape backup units, such as Colorado Memory Systems' (CMS) *Jumbo 250* and Tall-grass Technologies' *FS 150e*, can more

than double their storage capacity. Using StacPack, low-end *DC-2000* tapes, which normally hold only *40* Mbytes of data, can store up to *80* Mbytes on a single tape. File server owners can pack away *250* Mbytes on *DC-2120* tapes that can otherwise manage only *120* Mbytes.

Stac's method of data compression avoids the disk-bound penalties of most DOS software, but it still slows system performance due to the stealing of clock cycles. Despite this, Stac's software speeds backups since the time lost by compressing files is more than made up by the time gained in writing smaller amounts of data to tape [Ref.12].

**3. Compress**

*a. MS-DOS Ported Compress*

This is the MS-DOS ported version of UNIX 'compress', by Tsai, which uses adaptive Lempel-Ziv coding. The original UNIX 'compress' utility was written by S. W. Thomas, J. Mckie, S. Davies, K. Turkowski, J. A. Woods, and J. Orost[Ref.15]. COMPRESS is a *16*-bit LZW implementation in UNIX operating systems. The PC implementation that uses *16* bits takes up about *500K* of RAM [Ref.21].

*b. Modified Lempel-Ziv*

'Compress' uses the modified Lempel-Ziv algorithm. Common substrings in the file are first replaced by *9*-bit codes, *257* and up. When code *512* is reached, the algorithm switches to *10*-bit encoding and continues to use more bits

until the limit specified by the **-b** flag is reached (default *16*). The bits must be between *9* and *16*. The default can be changed in the source to allow 'compress' to be run on a smaller machine. After the bits limit is attained, 'compress' periodically checks the compression ratio. If the ratio is increasing, 'compress' continues to use the existing code dictionary. However, if the compression ratio decreases, 'compress' discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next block of the file. How much each file is compressed depends on the size of the input, the number of bits per code, and the distribution of common substrings [Ref.6]. Typically, text such as source code or English is reduced by *50-60%* [Ref.10]. Compression is generally much better than that achieved by Huffman coding or adaptive Huffman coding, and takes less time to compute [Ref.6].

## 4. ARJ221A

### *a. ARJ Evolution*

ARJ version 2.21a is written by Robert K Jung. It uses the LZ77 brute force hashing algorithm that outperforms all other LZ77 algorithms [Ref. 14]. ARJ is influenced by the design of LHARC written by H. Yoshizaki. The early version of ARJ also adapt the idea from AR001 of H. Okumura and some portion of ARJ is derived from AR source code [Ref.14].

*b. General Feature of ARJ*

ARJ is prototyped in ANSI C and only uses ANSI C standard libraries. The MS-DOS production of ARJ has functions of compression, extraction, CRC, and output routines (in assembler). For compressing, ARJ requires approximately *282* kbytes plus the memory necessary to store all of the path names to be archived when using the default compression method. For extracting, ARJ requires approximately *166* kbytes plus. There is no limitation on the number of files that can be stored in one archive. Examining the options of ARJ, one may find *4* methods. Different methods come from the emphasis among compression ratio and execution speed.

The default input is a binary mode but one may set the option to input text files for slightly better size reduction. If one use the 'text' mode for non-text files, ARJ will prematurely stop input if it finds an embedded EOF character (CTRL Z). This may produce a loss of data on binary files. The file type "text" is only needed for future cross platform transfers of ARJ archives. It enables ARJ to extract text files to the host file system with the text new line sequence that is correct for that operating system. This mode may produce slightly better size reduction, but extraction of files compressed in text mode is significantly slower than the extraction of binary files. In looking for *8*-bit non-text data, ARJ will look at the first *4096* bytes of the input file. If ARJ finds any *8*-bit data, it will automatically backtrack

and switch to binary mode for that particular file. In addition, at the end of compressing the input file, if ARJ finds that the input file size is not greater than 75 percent of the binary file size (size on disk), ARJ will report an error for that input file and increment the error count. This helps avoid the problem of accidentally compressing executable files with the text mode which results in lost data. The original file size reported by the "l" and "v" commands is the actual number of bytes inputted during text mode compression. This is usually the MS-DOS file size minus the number of carriage returns in the file since C text mode strips a file of carriage returns [Ref.14].

ARJ provides the capability of multiple volume archives. In other words, it can archive files directly to diskettes no matter how large or how numerous the input files are. It is possible to archive a 10 megabyte file to several diskettes and to recover the file directly from the diskettes. Other archivers, however, require that one compress the large file to hard disk or large RAM drive and then slice the compressed file to fit on diskettes. Recovering the original files involves reassembling the compressed file on the hard disk from the diskettes and then extracting the original files from the reassembled compressed file. This feature makes ARJ especially suitable for distributing large software packages without the concerns about fitting entire files on one diskettes. ARJ will automatically split files when necessary

and will reassemble them upon extraction without using any extra disk space [Ref.3].

The ARJ archive data structure with its header structure and *32* bit CRC code provide archive stability and recovery capabilities. This software also provides a security envelope facility by way of "lock" ARJ archives. A "locked" ARJ archive cannot be modified by ARJ. This provides some level of assurance to the user receiving a "locked" ARJ archive that the contents of the archive have not been tampered with. Data integrity checks contribute to the security of the ARJ "lock" [Ref.3].

**5. LHA213**

*a. New Static Huffman Coding*

This is a revised version of LH113c.exe, by H. Yoshizaki, an archiver which was rather slow in execution but tight in compression ratio. This LHA software employs new static Huffman coding instead of older dynamic Huffman coding and is faster than LH113c in decompressing but requires more memory than LH113c introduced by K. Okubo. This has been known as 'LHARC' since it was introduced in 1989 [Ref.3].

*b. General Feature of LHA*

LHA was chosen over runner-up ARJ because the header it attaches to its self-extracting module requires only *1.9* Kbyte of RAM, and is highly customizable. That means the SFX has features that make it especially helpful for users

distributing software. If one restricts the type of compression used, PKZIP's *2.6* Kbyte is competitive, but otherwise, the overhead in competing programs is *3* times as great or more. LHA requires *384K* plus the RAM [Ref.3].

This technique also is set so as not to compress for the files with extensions, .ARC, .LZH, .LZS, .PAK, .ZIP, .ZOO, which are partially or fully compressed already.

**6. PAK251**

*a. Distilling and Crushing*

This software uses the compression type of 'Distilled' and 'Crushed' among *12* compression types: Crunched, Squashed, Shrunk, Crushed, Imploded, Distilled ... 'Distilled' employs the Huffman coding and Sliding Window (LZ77) while 'Crushed' employs Lempel-Ziv algorithm.

*b. General Feature of PAK*

PAK is intended as a replacement for ARC by System Enhancement Associates and PKARC and PKZIP by Philip Katz [Ref.15]. While PKZIP 1.0 files are roughly comparable in size to PAK files, PAK supports multiple compression, more archive formats and features. PAK creates and modifies archive files which have the .PAK, .ARC, or .ZIP extension. Files in an archive retain all of the information they had in the directory, such as name, size, and date. In addition, each file in an archive has a calculated CRC number, which assures the detection of damage after events such as file transmission

via modem. The basic format of PAK has *1* byte of marker, *1* byte of version, *13* bytes of name, *4* bytes of size, *2* bytes of data, *2* bytes of time, *2* bytes of CRC, and *4* bytes of length. Basic archives end with a short header, containing just the marker (*26*) and the end of file value (*0*) [Ref.15].

PAK has a wide array of extra features that includes comment writing, password protection, and a security envelope. PAK's optional command shell makes use of pop-up windows [Ref.15], which still is the most pleasing interface among any of the six programs evaluated here.

# IV. PERFORMANCE ANALYSIS OF COMPRESSION SOFTWARE

## A. EXPERIMENTAL SETUP

We define the compression ratio as the size of compressed file divided by the size of original file such that the smaller the compression ratio, the better the performance. Some software may use different measures for indicating the compression effectiveness such as 'SF (Stowage Factor)' which is the percentage of the reduction in file size by compression [Ref.22]. In archiving, the total Stowage Factor is the stowage factor for the archive as a whole, not counting archive overhead. In this thesis, however, we use the compression ratio defined above.

### 1. How Files Are Tested

There are many ways to classify data files. Generally speaking, one can classify data files into ASCII type and binary type. An ASCII file is a data or text file that contains only characters coded from the standard ASCII printable character set. A binary file is generated in machine language form and ready to be executed by the CPU. Binary files cannot be transmitted by protocols that handle pure ASCII text.

This thesis classifies the data files into Text, Executable, dBASE, and Image files since this classification

meets the practical need of data management and transmission, especially in the military environment [Ref.24].

There are possibly many different types or formats in Image files: scanned picture, black-and-white image, color image, etc. In the compression analysis, however, they are all classified as Image type.

For comparison, *3* compression methods: PKZIP, StacPack, and Compress, the ported version of Compress in UNIX to DOS, and *4* archiving methods: ARJ221A, LHA213, PKZIP, and PAK251, were examined. Note that the *4* archiving techniques also contain the function of compression. For a wide range comparison, files sized from *500* bytes to *1* megabytes were collected. The file sizes spanned over *0.5K, 1K, 1.8K, 3K, 5K, 8K, 13K, 20K, 40K, 70K, 120K, 190K, 300K, 500K,* and *800K*. The margin of each size is *±20%* which made for a relatively even and wide spread range. To test data compression packages, a collection of as many files as possible were gathered; however, *5* sample files for each of the *15* representative sizes constituted each file type.

The files are collected from the computers at NPS. They are mainly files of personal computers, DOS operated, though some were from VAX, and SUN workstations. The total size of each type of file ranges from *4* megabytes up to *10* megabytes. In any event, a compression or archiving software was needed to reduce the time and effort required to collect and manage those files.

Experiments were run on a *33*-MHz IBM (Compatible) Desktop *486* with *8* MB of extended RAM and a *100* MB hard disk. The hard disk was formatted under MS-DOS 4.01. Sample files were stored on hard disk. Furthermore, these experiments were conducted in the program's native(default) mode.

## 2. Sample Files Classification

**Text files** include word processing documents, batch files and source language programs and are usually ASCII files as they contain only letters, digits and symbols. Most of the files are from mathcad [Ref.25], matlab [Ref.26], wp51 [Ref.28], PSpice [Ref.27], C++ [Ref.31]. Note that although text files are generally human-readable, the compressed files are generally not.

**Executable files** include machine language programs ready to be loaded and executed in the computer. These executable (binary) files may have some ASCII text in them as string constants. A total of more than *8* megabytes of executable files were obtained. There files are generally found with file extension .EXE or .COM. In contrast with .COM file, which is designed to work only in specific memory locations, .EXE files are designed as relocatable files and can reside in any memory locations. Most of the executable files were collected from DOS operating computers. They can be compressed with slightly larger (worse) ratios than text

files. Moreover, they need 3 times longer processing time than that required by ASCII text files.

**A database** is a collection of interrelated files that are created and managed by a database management system(DBMS). In the following discussion the word 'database' implies DBASE IV because it is the most widely used database system for personal computers, and its programming language and file formats have become industry standards. Additionally, DBASE is widely used in the U.S. Navy; therefore the compression effectiveness of dBASE files should be studied separately. Database files are usually not ASCII files since they contain numbers in integer or floating point forms and many control codes for tabulating purpose.

Due to the difficulty of obtaining a sufficient number of dBASE files, some files are acquired from the example files of dBASE IV, some files are purposely composed for different sizes, and some are obtained through the **ftp** (file transfer protocol) over internet from public domains.

**Computer graphics** and image processing applications create and process digital images. Images can be generated or sensed before they are stored in computers. For storing and maintaining pictures in a computer, images are represented in either vector graphics or raster graphics. When circuits are drawn in CAD (Computer Aided Design), vector graphics is used. As one draws, each line of the image is stored as a vector (two end points on a two dimensional matrix). Vector graphics

maintain the image as a series of lines. Unlike vector graphics, raster (binary) graphics is used when objects are "painted" on screen or are scanned, typically from *16* to *256* levels of gray levels, into the computer. It is similar to television where the picture image is made up of dots (pixels).

The *10* megabytes image files are collected including CAD files [Ref.29], drawperfect sample files [Ref.33], business graphics files which yield graphics-like bar or pie charts, or scatter diagrams, files from commercial games, and some Black/White and some colored images. Like Executable files, Image files may include some text descriptions that provide charts, tables, and special characters. Through **ftp** some larger sized files (above *300K*) were downloaded from various universities and institutions.

## B. EXPERIMENTAL RESULT ANALYSIS

### 1. Text Files

Fig. 5 shows the average compression ratios of PKZIP, StacPack, and Compress on collected text files. PKZIP ranks best when applied to text files.

Text files in the range of [*10K*, *100K*] benefit the most since the compression ratios are lower than those of other files sizes. PKZIP stood out as *21.4%* at *190K*. Looking at each sample file (See Appendix A), one finds a PSpice library file sized *135K* was compressed to *7%* of its original

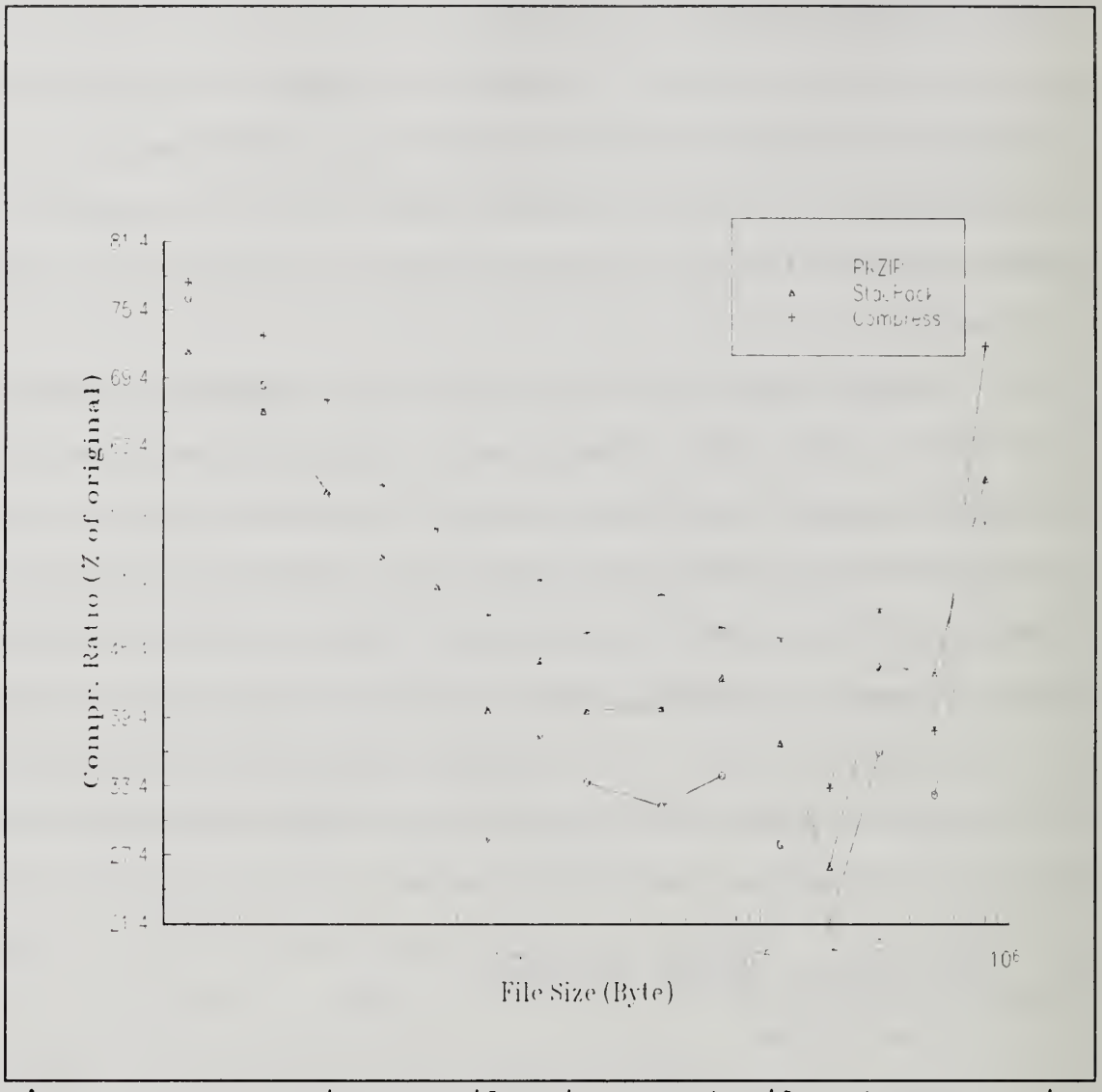


**Fig. 5.** Compression vs File Size, Text Files ( Compression Only ).

size using PKZIP. This is no surprise since there are many blanks in the library file. PKZIP's average ratio was *36%*, StacPack was *43%*, and Compress was *50%*.

Fig. 6 is the comparison among *4* packages mentioned in section IV.A. One observes little difference from the lines.

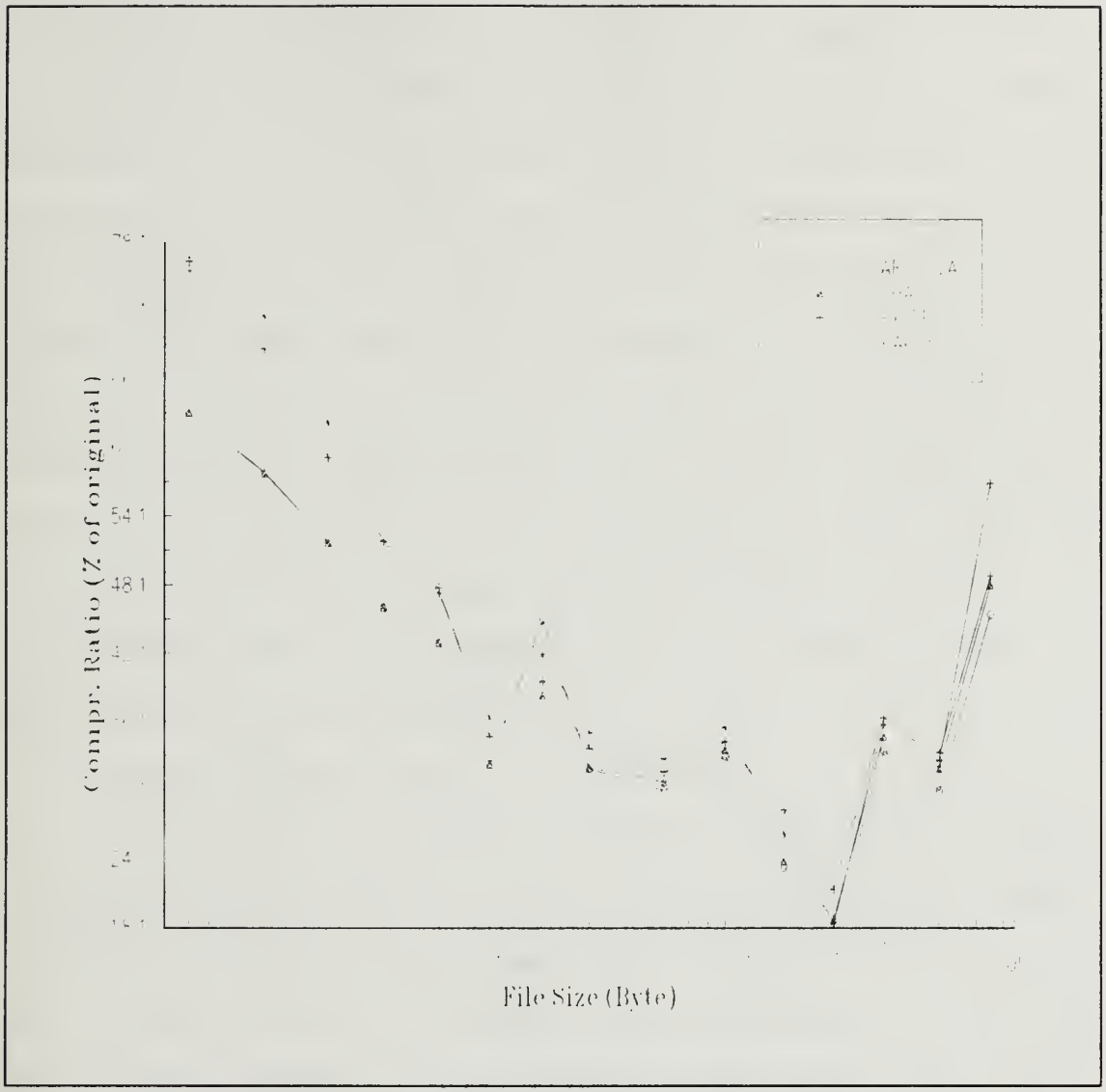


**Fig. 6.** Compression vs File Size, Text Files ( Compressed & Archived ).

However, ARJ221A stood out as the best and LHA213 was a close second. Good compression ratios are spread evenly between *10K* and *200K* which is consistent with the findings in Figure 5. Notably, for small size files, one does not find good ratios because the overheads of the software packages are too overwhelming. Comparing with PKZIP, the PSpice file at *135K*

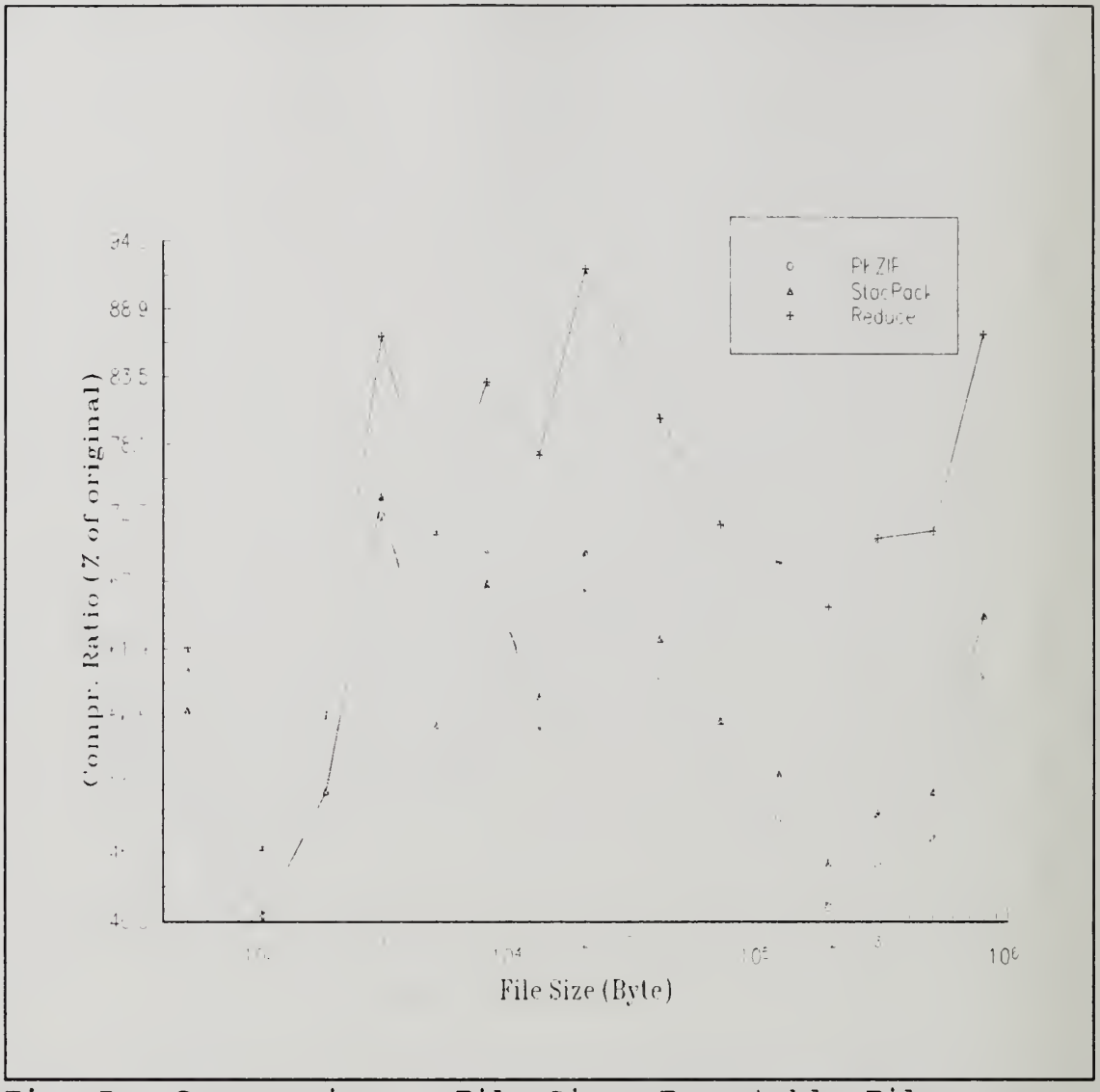


**Fig. 7.** Compression vs File Size, Executable Files ( Compression Only ).

was compressed to *6.2%* by ARJ and LHA. The overall ratios of packages were *31%*, *32%*, *36%*, and *34%* for ARJ, LHA, PKZIP, and PAK251, respectively.

**2. Executable Files**

Fig. 7 and Fig. 8 compare the compression ratios of Executable files among *6* software packages. The curves show

much more peaks and troughs than text files. However, the size ranges between *30K* and *300K* is a most stable range with better compression ratio than the other ranges. As sample size grows in archiving, ARJ is better than LHA, and PKZIP and PAK251 are tied. Additionally, one recognizes that 'Compress' does not perform well for .EXE file compression. Notably, PKZIP compressed PKUNZIP.EXE file to *77%*, ARJ and LHA to *74%*, but Compress shows an expansion or *102%* of its original file. PAK251's *7.6%* ratio for a *1.1K* gen41.exe is the smallest ratio. Average ratios of each package was *51%* for PKZIP, *56%* for StacPack, *76%* for Compress, *48%* for ARJ221A, *49%* for LHA213, and *49%* for PAK251.

**3. dBASE Output Files**

Fig. 9 and 10 show the curves that are somewhat linear as the size grows. That is because when the file size grows, the amount of overhead or format has little difference with that of small size file. Sample sizes between *20K* and *500K* show the most useful range of dBASE Output File size to get the smallest value of compression ratio. In Fig. 9, after *10K*, Compress is approximately *10%* better than StacPack, and follows closely to PKZIP. In Fig.10, PAK251 also outperforms over PKZIP after *20K*.

The smallest ratio from dBASE Output File is *12.3%* at *13K* of 'quad.dbf'. This file contains accounting information of personal names and addresses. Average compression ratios of

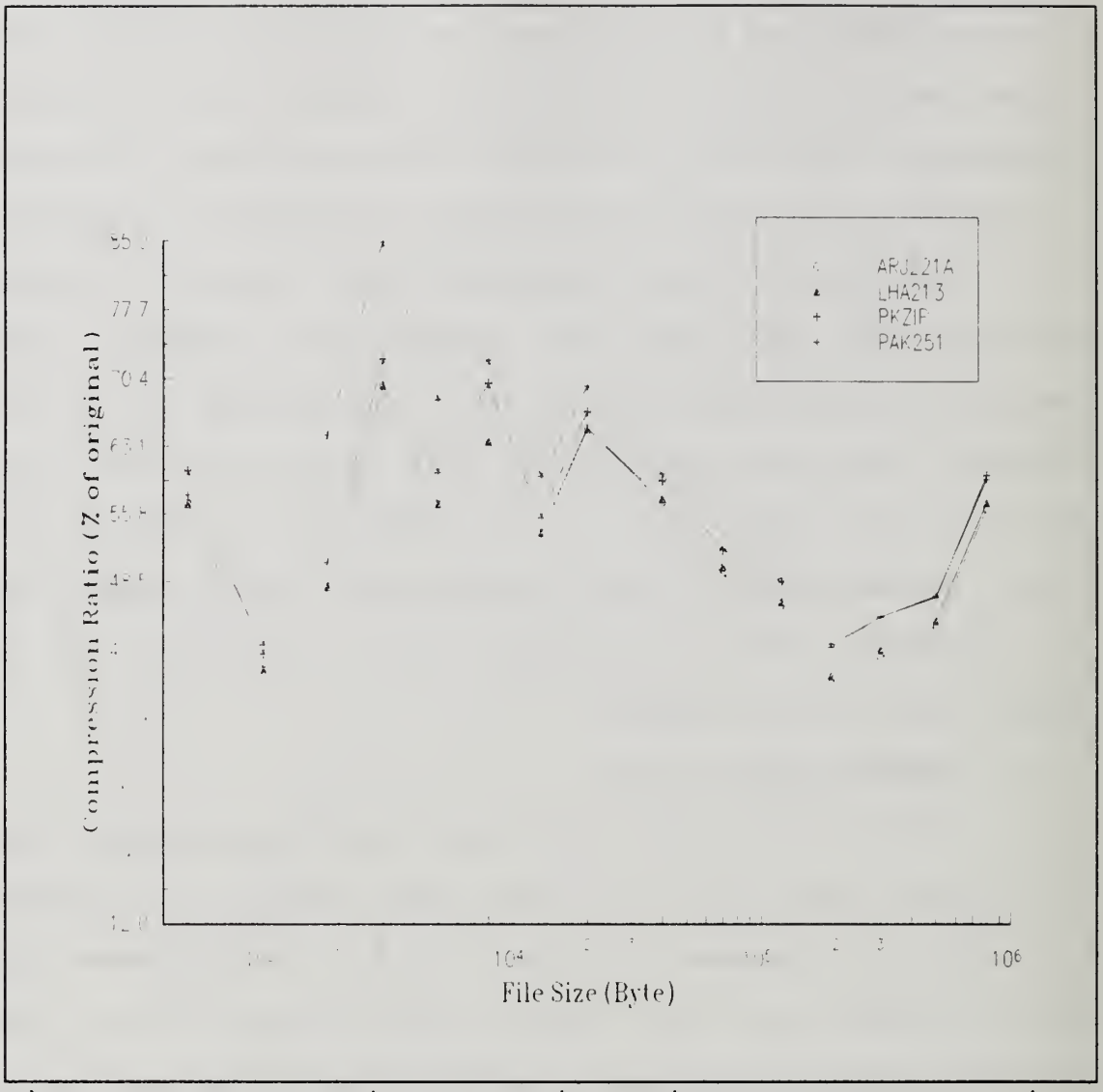


**Fig. 8.** Compression vs File Size, Executable Files (Compression & Archived).

dBASE files are *22%* for PKZIP, *29%* for StacPack, *24%* for Compress, *18%* for ARJ, *18%* for LHA, and *19%* for PAK251.

**4. Image Files**

Curves in Fig. 11 and 12 show V-shaped plots except for abrupt jumps at *70K* range. This might be because of 'bv.sr' and 'bfg.sr', which are Black/White normal pictures.

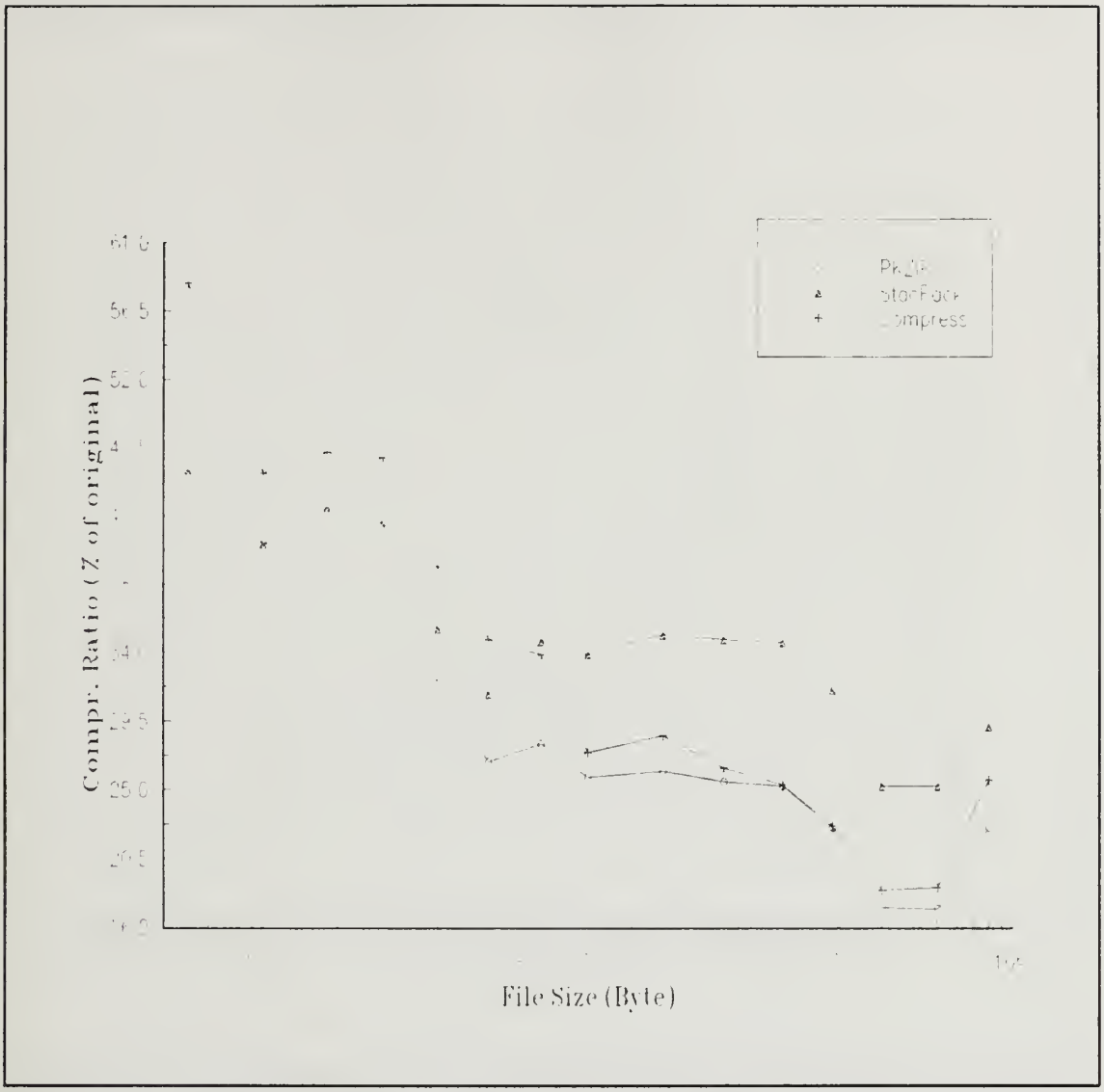


**Fig. 9.** Compression vs File Size, dBASE Output Files ( Compression Only ).

Except for the 70K cases the results show that the files size between *10K* and *100K* are benefit most from the compression.

Graphics users must note that some image files are resistant to the compression algorithms. For instance the gray-scaled .GIF image files have *100%* to *132%* compression ratios. This indicates there is some overhead generated by the

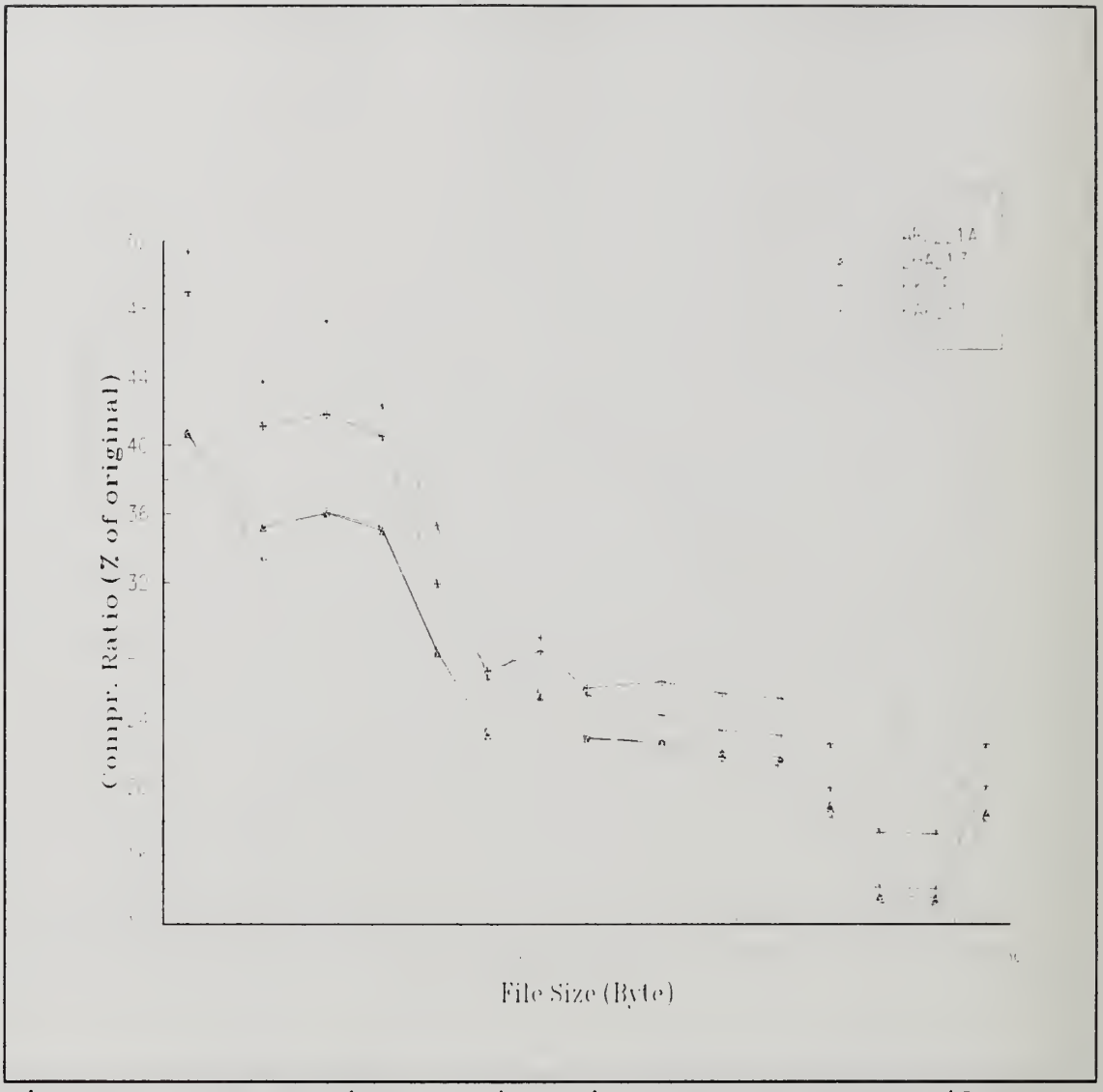


**Fig. 10.** Compression vs File Size, dBASE Output Files ( Compressed & Archived ).

software package. If one needs to compress those files, it is necessary to change the format from .GIF to .PCX or to whatever is compressible. It is noted that one can convert .GIF to .PCX format (with some expansion) and then compress the .PCX files. By doing this one can have a net compression ratio of less than *1*.

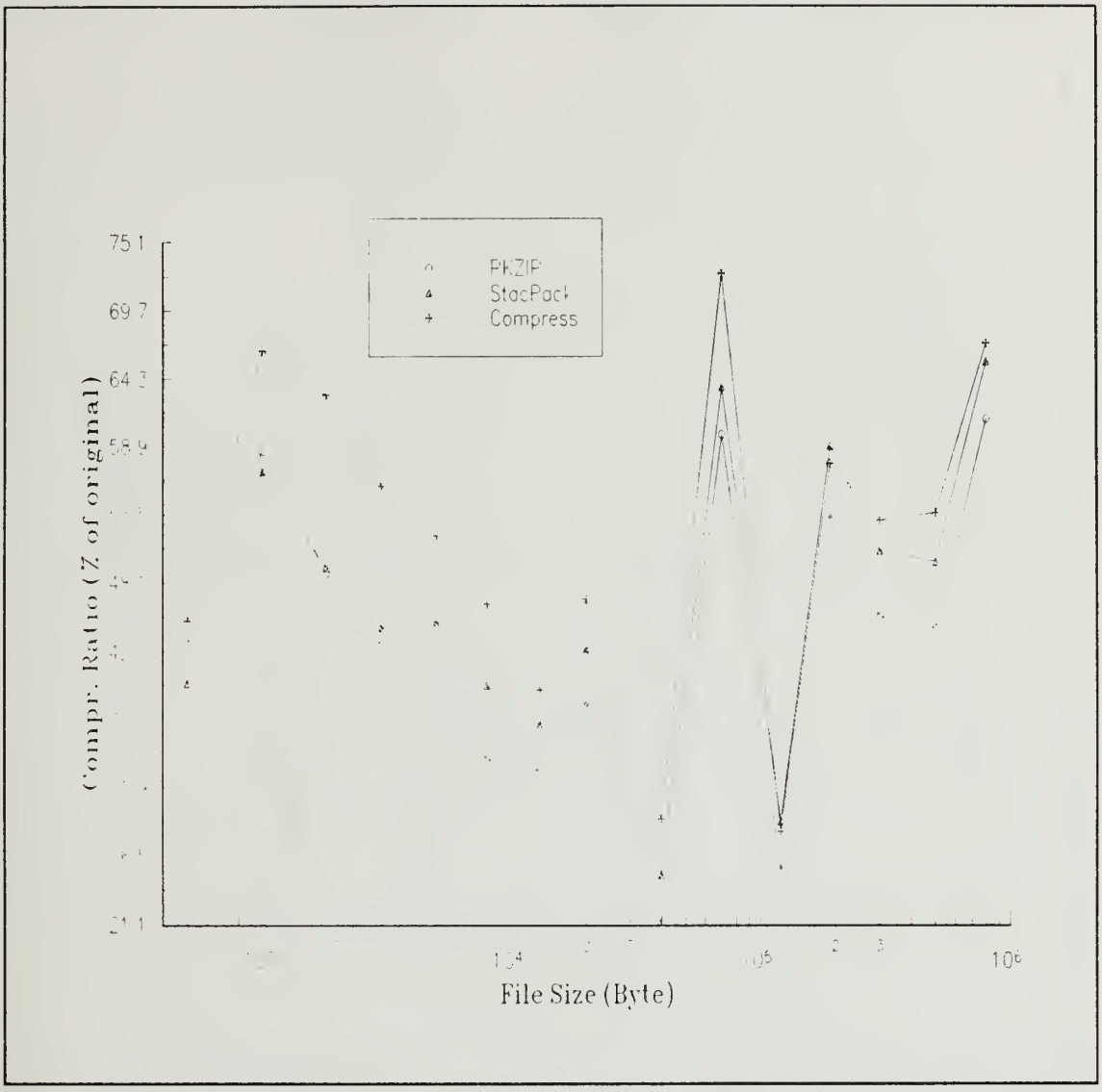


**Fig. 11.** Compression vs File Size, Image Files ( Compression Only ).

ARJ and LHA remain as the best compression software in compressing image files. 'Scree.rf' at *40K* has a compression ratio of *6%* which is the best from the experiment by ARJ and LHA. Each of ARJ and LHA has its own favorites; for example, 'bdy2.cbd' at *190K* by ARJ was *6%*, but *48%* by LHA. The overall compression ratios were *51%, 56%, 58%, 46%, 47%*, and *52%* by

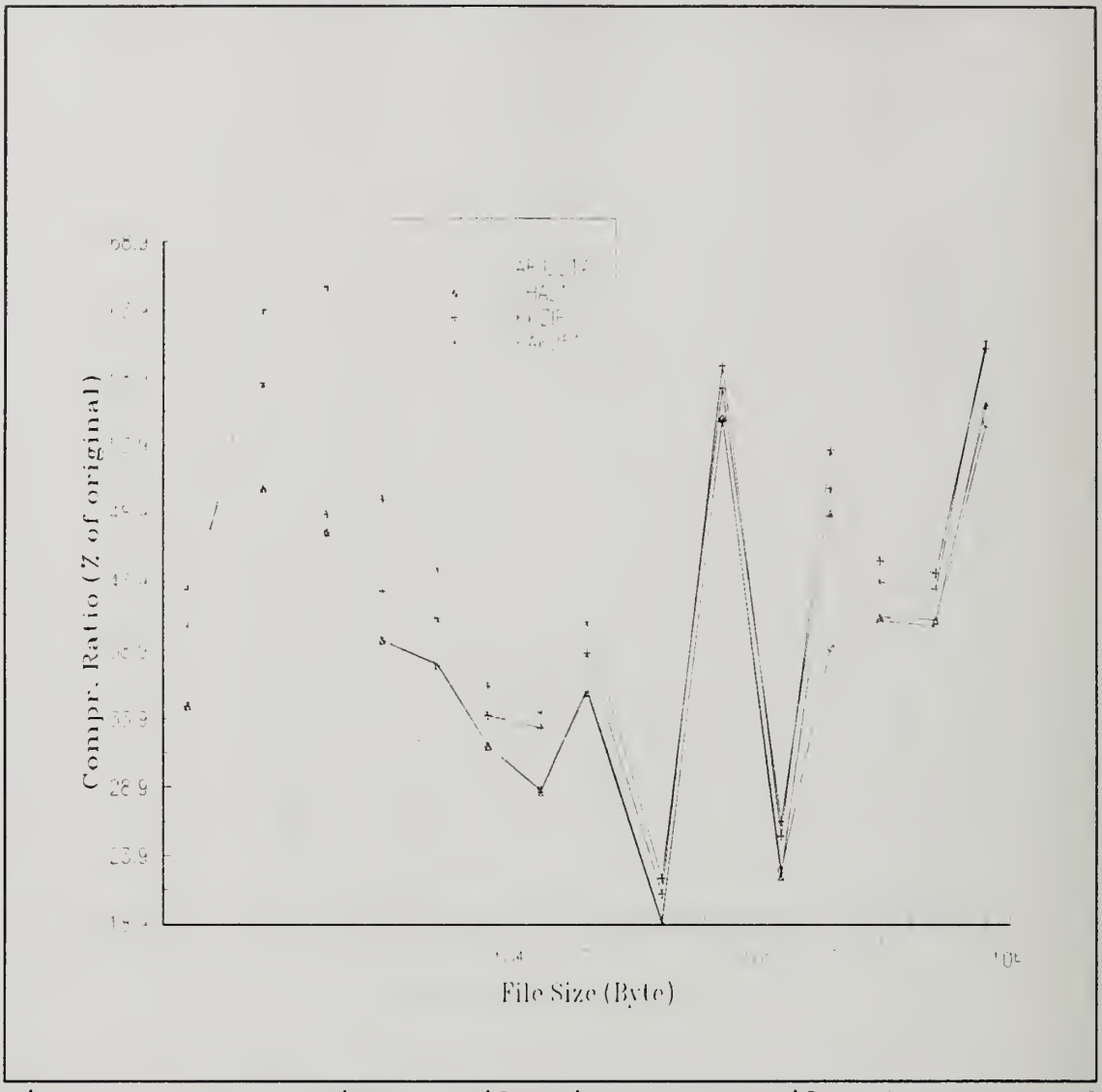


**Fig. 12.** Compression vs File Size, Image Files ( Compressed & Archived ).

PKZIP, StacPack, Compress, ARJ221A, LHA213, and PAK251, respectively.

### 5. Overall Performance Analysis

'Compress' shows the worst capability in Executables, but better than or close to StacPack in dBASE and Image files. PKZIP had the same average compression ratio in Image and

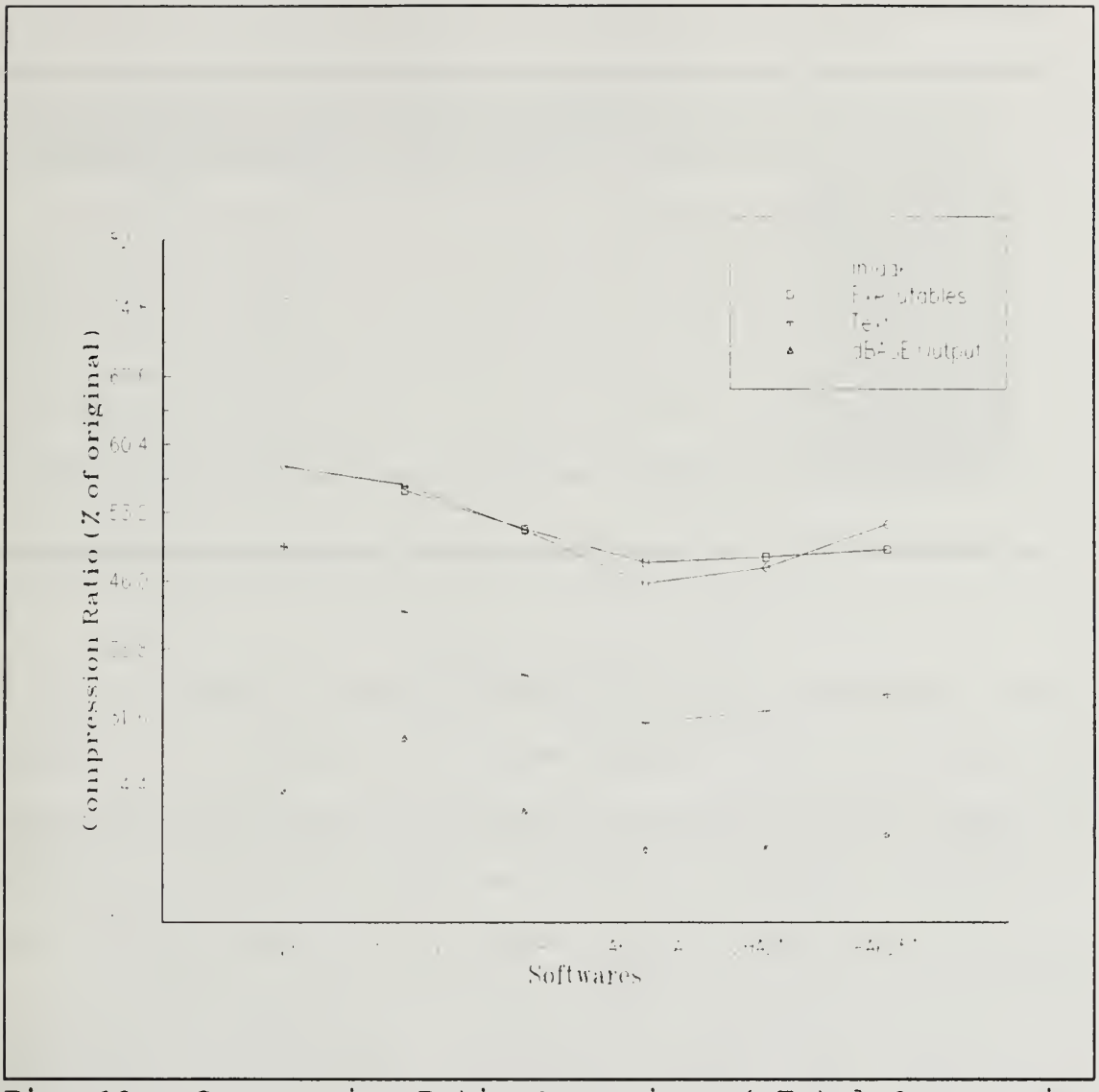


**Fig. 13.** Compression Ratio Comparison ( Total Compression of Each File Type ).

Executable files. Besides, one has to recognize that the .ZIP file format is the current standard in the data compression world. ARJ and LHA have kept steady low compression ratios in most kinds of file. ARJ proved slightly more effective on every

**Table V** Compression Ratio Comparison

| % | Text | Execute | dBASE | Image |
|---|---|---|---|---|
| PKZIP | 36.0 | 51.4 | 21.8 | 51.3 |
| StacPack | 42.7 | 55.5 | 29.4 | 58.1 |
| Compress | 49.6 | 76.2 | 23.9 | 58.1 |
| ARJ221A | 34.9 | 47.9 | 17.7 | 45.7 |
| LHA213 | 32.2 | 48.5 | 18.0 | 47.4 |
| PAK251 | 34.0 | 49.3 | 19.3 | 51.9 |

type. However, they are only *1.3%* in Text, *0.6%* in Executables, *0.3%* in dBASE, and *1.7%* in Image files. LHA gets the nod over ARJ because the header it attaches to its self-extracting modules is both the smallest among the six programs (*1.9K*) and the one with the most potential for customization. If we use < to indicate the relative compression ratios, then *ARJ < LHA < PAK < PKZIP < StacPack < Compress*. In other words, ARJ outperforms the others. Using the self-extracting technique allows the sending of compressed files to a party who does not have any utility to decompress them.

The files compressed by PKZIP were mostly 'imploded' which employed LZ77 and Shannon-Fano coding. With this in mind, considering the algorithms of good performing software packages, one can conclude that LZ77(SW), Huffman, and Shannon-Fano create the least compression ratio.

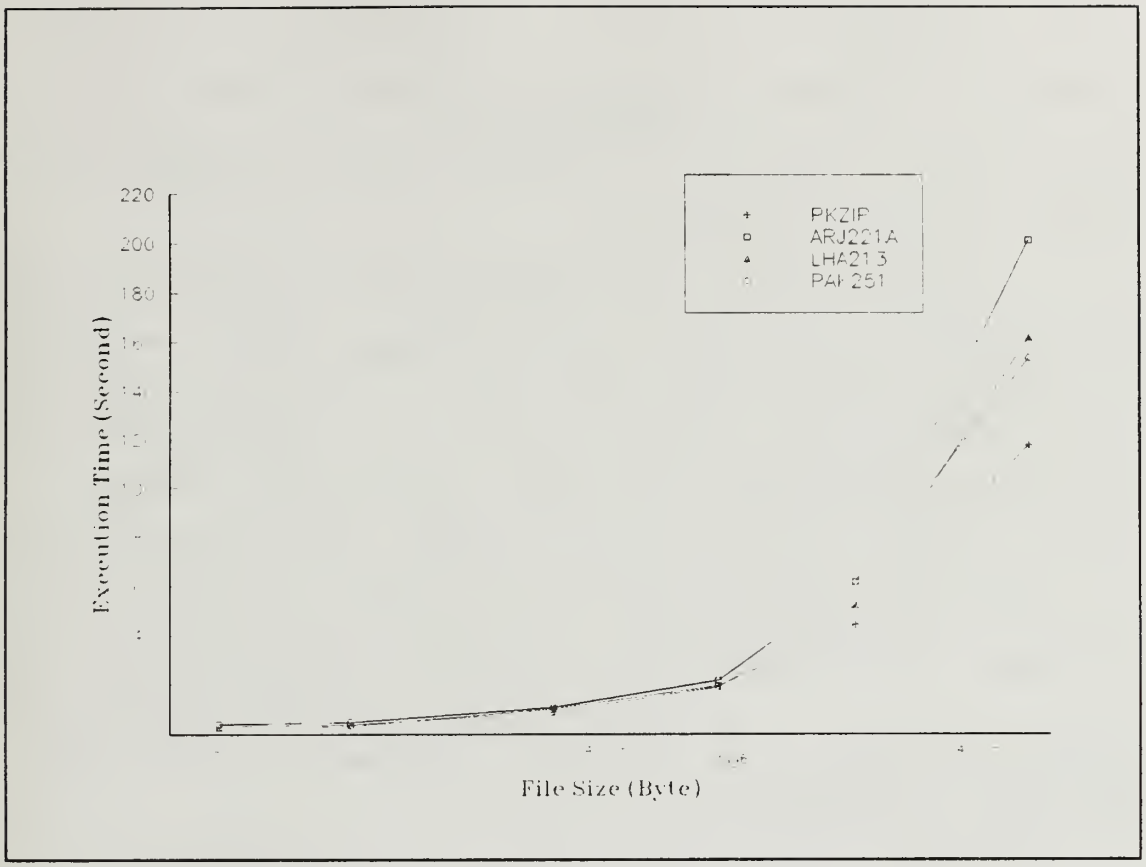


**Fig. 14.** Execution Time Comparison (Compressed & Archived).

Figure 13 shows that the dBASE files can be compressed the most in comparison to other file types. Notice that the binary files, Executables and Image files, have the highest compression ratios.

Although the differences are slight, some products outperformed others in compressing particular types of files. ARJ was best at compressing ASCII and executable files, while LHA realized the most out of the graphics formats. PAK251 is better than PKZIP in compression ratio except Image files, although the difference is a mere *0.6%*.

Table V shows the general compression ratio for *4* file types and *6* packages. As one see, ARJ ranks at the top in all file types, and Compress the last. It is also shown in Figure 14 for clarity of comparison.

Figure 14 shows the execution time of 4 archivers. One cannot see big difference among softwares up to 1 Megabytes. However, PKZIP on a *33*-MHz *486* with a hard disk of *18*ms access time took *44* seconds to compress and archive *2* Mbytes of *7* sample files. In the same environment, ARJ took *17* seconds more and LHA took *8* seconds more than that of PKZIP. On the average, PKZIP is the fastest product. LHA and ARJ, the best compressors, still lagged behind the leader in speed. Details are shown in Appendix C.

# V. CONCLUSION

Archiving and data compression utility programs allow users to store data files in a highly compressed form, which conserves storage space and improves telecommunication services. Archiving utilities also permit groups of files to be stored together in a single 'archive' file. Single files are easier to move, copy, store and manage than are ad-hoc collections of individual files [Ref.3]. There is no distinction between compression and archiving for softwares that provide archiving only.

Efficient information queries on archived and/or compressed files without unbundling the entire file systems is one important area for further research.

It is believed that compression will play a greater role in the future of personal computers and data communication. This is particularly true in multi-media applications where large amount of information have to be transferred and stored. However, that may require irrecoverable compression.

While data compression is not appropriate for every application, nearly *30* years of research on the subject has demonstrated that there are ample areas for research. It is valuable in data processing for efficient data transfer and storage.

As all the techniques have developed, we see now that data compression has become a part of routine data processing and communications. There are still many problems related to data compression that remains to be solved. For example, error detection and error correction are not incorporated in most software packages. A major use of data compression today is in communication systems. Compressing a message reduces the time and cost of sending it by an amount often equal to the compression ratio. Several popular softwares for data compression and archiving have been investigated and applied to files collected at NPS. The results show, in general, PKZIP is the fastest and ARJ221A has the best compression ratio. Therefore ARJ221A archives relatively the best. The details are reported in Chapter IV.

## APPENDIX A. RESULT OF EXPERIMENT FOR COMPRESSION SOFTWARE

Table 1 COMPRESSED, TEXT FILES < Fig.5 >

** For various sizes of Text Files, Compressed only

| File Size | Text | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|
| 0.5K | shutt.mcd | 555 | 418 | 75.3 | 394 | 71.0 | 423 | 76.2 |
| 400 | oilri.mcd | 554 | 416 | 75.1 | 393 | 70.9 | 421 | 76.0 |
| -600 | spira.mcd | 640 | 489 | 76.4 | 462 | 72.2 | 501 | 78.3 |
| | cond .m | 446 | 357 | 80.0 | 331 | 74.2 | 353 | 79.1 |
| | dec2h.m | 555 | 419 | 75.5 | 395 | 71.3 | 444 | 80.0 |
| | Avg | 550 | 420 | 76.4 | 395 | 71.8 | 428 | 77.8 |
| 1K | feath.m | 1207 | 785 | 65.0 | 763 | 63.3 | 856 | 70.9 |
| 800 | anhar.mcd | 1025 | 735 | 71.7 | 714 | 69.7 | 774 | 75.5 |
| 1200 | polar.mcd | 809 | 593 | 73.3 | 569 | 70.4 | 618 | 76.4 |
| | hex2n.m | 1053 | 689 | 65.4 | 669 | 63.6 | 740 | 70.3 |
| | expm1.m | 804 | 563 | 70.0 | 544 | 67.7 | 593 | 73.8 |
| | Avg | 980 | 673 | 68.7 | 652 | 66.5 | 716 | 73.1 |
| 1.8K | bode .mcd | 2258 | 1367 | 60.5 | 1381 | 61.2 | 1547 | 68.5 |
| 1440- | boole.mcd | 1455 | 880 | 60.5 | 878 | 60.4 | 1003 | 68.9 |
| 2160 | brake.mcd | 1947 | 1136 | 58.3 | 1154 | 59.3 | 1275 | 65.5 |
| | compf.mcd | 1528 | 926 | 60.0 | 915 | 59.9 | 1041 | 68.1 |
| | erf .m | 2062 | 1161 | 56.3 | 1166 | 56.6 | 1370 | 66.4 |
| | Avg | 1850 | 1094 | 59.1 | 1099 | 59.4 | 1247 | 67.4 |
| 3K | anten.doc | 2737 | 1564 | 57.1 | 1609 | 58.8 | 1678 | 61.3 |
| 2400- | mks.mcd | 3772 | 1864 | 49.4 | 1961 | 52.0 | 2140 | 56.7 |
| 3600 | besse.m | 2426 | 1331 | 44.9 | 1352 | 55.8 | 1487 | 61.3 |
| | bilin.m | 3076 | 1570 | 51.0 | 1604 | 52.2 | 1928 | 62.7 |
| | cplxp.m | 3021 | 1474 | 48.8 | 1523 | 50.4 | 1774 | 58.7 |
| | Avg | 3006 | 1561 | 51.9 | 1610 | 53.6 | 1801 | 59.9 |
| 5K | read1.doc | 4259 | 2258 | 53.0 | 2391 | 56.2 | 2567 | 60.3 |
| 4K- | inst.doc | 4029 | 1988 | 49.3 | 2088 | 51.8 | 2279 | 56.6 |
| 6K | readm.txt | 5594 | 2485 | 44.4 | 2704 | 48.3 | 3050 | 54.5 |
| | cgs.mcd | 4383 | 2110 | 48.1 | 2246 | 51.3 | 2420 | 55.2 |
| | direc.mcd | 5112 | 2324 | 45.5 | 2503 | 49.0 | 2776 | 54.3 |
| | Avg | 4675 | 2233 | 47.8 | 2386 | 51.0 | 2618 | 56.0 |
| 8K | stmed.msg | 7900 | 3033 | 38.4 | 3483 | 44.1 | 4126 | 52.2 |
| 6400- | redm.mcd | 7615 | 3547 | 46.6 | 4096 | 53.8 | 4357 | 57.2 |
| 9600 | bench.m | 7377 | 2615 | 35.4 | 2882 | 39.1 | 3841 | 52.1 |
| | spi2.dat | 9449 | 2236 | 23.7 | 2743 | 26.2 | 3172 | 33.6 |
| | fload.c | 8727 | 2834 | 32.5 | 3261 | 37.4 | 4390 | 50.3 |

** For various sizes of Text Files, Compressed only

| File Size | | Text | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| | Avg | 8213 | 2330 | 28.4 | 3293 | 40.1 | 3977 | 48.4 |
| 13K | api.doc | 15240 | 5815 | 38.2 | 6960 | 45.7 | 7939 | 52.1 |
| 10.4- | textb.doc | 15429 | 6919 | 44.8 | 7320 | 42.4 | 9835 | 63.7 |
| 15.6K | read.doc | 15443 | 5774 | 37.4 | 7177 | 46.5 | 7459 | 48.3 |
| | remez.m | 15407 | 4472 | 27.0 | 5227 | 33.9 | 6544 | 42.5 |
| | read3.doc | 12006 | 4853 | 40.4 | 5990 | 49.9 | 6074 | 50.6 |
| | Avg | 14705 | 5567 | 37.9 | 6535 | 44.4 | 7570 | 51.5 |
| 20K | thesi.doc | 17408 | 6207 | 35.7 | 7385 | 42.4 | 9232 | 54.2 |
| 16K- | arrow.doc | 21582 | 10226 | 47.4 | 10699 | 49.0 | 16224 | 75.3 |
| 24K | cshel.doc | 24911 | 7775 | 31.2 | 10106 | 40.6 | 9620 | 38.6 |
| | redu.c | 21931 | 4981 | 22.7 | 6782 | 30.9 | 7201 | 32.8 |
| | spi1.dat | 21477 | 7008 | 32.6 | 7985 | 37.2 | 7918 | 36.9 |
| | Avg | 21461 | 7239 | 33.7 | 8591 | 40.0 | 10039 | 46.8 |
| 40K | chara.doc | 42223 | 13881 | 32.9 | 16665 | 39.5 | 23361 | 49.6 |
| 32K- | matla.hlp | 50425 | 20556 | 40.8 | 25277 | 50.1 | 26260 | 52.1 |
| 48k | setup.inf | 50014 | 12898 | 25.8 | 15882 | 31.8 | 24094 | 48.2 |
| | eval.lib | 52515 | 15614 | 29.7 | 21426 | 40.8 | 27348 | 52.1 |
| | parts.hlp | 33583 | 9424 | 28.1 | 12721 | 37.9 | 13855 | 40.9 |
| | Avg | 45752 | 14435 | 31.6 | 18394 | 40.2 | 22984 | 50.2 |
| 70K | holid.doc | 55584 | 31797 | 57.2 | 33152 | 59.6 | 48164 | 86.7 |
| 56K- | mcad.hlp | 53184 | 13639 | 25.6 | 19123 | 36.0 | 17183 | 32.3 |
| 84K | check.hlp | 52616 | 17631 | 33.5 | 23455 | 44.6 | 21757 | 41.4 |
| | util.doc | 79144 | 24687 | 31.2 | 32078 | 40.5 | 33434 | 42.2 |
| | class.doc | 55736 | 13952 | 25.0 | 19184 | 34.4 | 19592 | 35.2 |
| | Avg | 59253 | 20341 | 34.3 | 25398 | 42.9 | 28026 | 47.3 |
| 120K | qbasi.hlp | 130810 | 130810 | 100. | 122332 | 93.5 | 155921 | 119. |
| 96K- | anlg.lib | 138727 | 18629 | 13.4 | 61036 | 44.0 | 48669 | 35.1 |
| 144K | tex.lib | 131653 | 10137 | 7.7 | 16533 | 12.6 | 32684 | 24.8 |
| | thyri.lib | 135346 | 9409 | 7.0 | 16053 | 11.9 | 27136 | 20.0 |
| | lin.lib | 110682 | 14313 | 12.9 | 24944 | 22.5 | 36165 | 32.7 |
| | Avg | 129444 | 36660 | 28.3 | 48180 | 37.2 | 60115 | 46.4 |
| 190K | ssims.mdr | 212493 | 23513 | 11.1 | 37391 | 17.6 | 30370 | 14.3 |
| 152K- | eval2.dat | 159201 | 98536 | 61.9 | 101937 | 64.0 | 131712 | 82.7 |
| 228K | bipol.lib | 185420 | 25906 | 14.0 | 39172 | 21.1 | 46633 | 25.1 |
| | diode.lib | 158181 | 22716 | 14.4 | 30692 | 19.4 | 44552 | 28.2 |
| | pwr.lib | 184757 | 22377 | 12.1 | 28532 | 15.4 | 45859 | 24.8 |
| | Avg | 180010 | 38610 | 21.4 | 47545 | 26.4 | 59825 | 33.2 |
| 300K | quatt.hlp | 287589 | 104755 | 36.4 | 126345 | 43.9 | 141000 | 49.0 |
| 240K- | | | | | | | | |
| 300K | Avg | 287589 | 104755 | 36.4 | 126345 | 43.9 | 141000 | 49.0 |

** For various sizes of Text Files, Compressed only

| File Size | | Text | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| 500K | ridm.txt | 454374 | 147912 | 32.6 | 197406 | 43.4 | 173423 | 38.2 |
| 400K-600K | Avg | 454374 | 147912 | 32.6 | 197406 | 43.4 | 173423 | 38.2 |
| 800K | tchel.tch | 976250 | 555033 | 56.9 | 590872 | 60.5 | 704621 | 72.2 |
| 640K-960K | Avg | 976250 | 555033 | 56.9 | 590872 | 60.5 | 704621 | 72.2 |
| | Total | 4,067,706 | 1,463,707 | | 1,735,553 | | 2,015,783 | |
| | Ratio | 100 % | 36.0 % | | 42.7 % | | 49.6 % | |

Table 2 COMPRESSED, EXECUTABLE FILES < Fig.7 >

** For various sizes of Executable Files, Compressed only

| File Size | Execute | | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | isat.exe | 568 | 104 | 18.3 | 94 | 16.6 | 109 | 19.2 |
| 400- | chkri.com | 688 | 604 | 87.8 | 586 | 85.2 | 628 | 91.3 |
| 600 | rambi.com | 307 | 268 | 87.3 | 232 | 75.9 | 271 | 88.3 |
| | exet1.com | 413 | 413 | 100. | 396 | 96.1 | 413 | 100. |
| | fasto.exe | 680 | 215 | 31.6 | 207 | 30.5 | 224 | 32.9 |
| | Avg | 531 | 321 | 60.5 | 303 | 57.1 | 329 | 62.0 |
| 1K | egaep.com | 1006 | 665 | 66.1 | 655 | 65.2 | 739 | 73.5 |
| 800- | gen41.exe | 1125 | 118 | 10.4 | 120 | 10.7 | 132 | 11.7 |
| 1200 | loadf.com | 1131 | 607 | 53.7 | 601 | 53.2 | 699 | 61.8 |
| | prtsc.exe | 1176 | 419 | 35.6 | 416 | 35.4 | 468 | 39.8 |
| | Avg | 1110 | 452 | 40.7 | 448 | 40.3 | 510 | 45.9 |
| 1.8K | curso.com | 1452 | 1183 | 81.5 | 1175 | 81.0 | 1316 | 90.0 |
| 1440- | gen42.exe | 1477 | 176 | 11.9 | 184 | 12.5 | 216 | 14.6 |
| 2160 | 67ves.com | 1559 | 999 | 64.1 | 993 | 63.8 | 1189 | 76.3 |
| | runti.exe | 1590 | 758 | 47.7 | 766 | 48.2 | 811 | 51.0 |
| | dbase.exe | 1588 | 754 | 47.5 | 762 | 48.0 | 808 | 50.9 |
| | Avg | 1533 | 774 | 50.5 | 776 | 50.6 | 868 | 56.6 |
| 3K | egala.com | 2388 | 1152 | 48.2 | 1170 | 49.0 | 1678 | 61.3 |
| 2400- | more.com | 2618 | 2044 | 78.1 | 2058 | 78.6 | 2140 | 56.7 |
| 3600 | appen.exe | 2902 | 2902 | 100. | 3073 | 106. | 3574 | 123. |
| | setna.exe | 3174 | 1977 | 62.3 | 1977 | 62.9 | 2308 | 72.7 |
| | astcl.com | 2557 | 1796 | 70.2 | 1817 | 71.1 | 2111 | 82.6 |
| | Avg | 2728 | 1974 | 72.4 | 2019 | 74.0 | 2362 | 86.6 |
| 5K | edit.exe | 4837 | 3654 | 75.5 | 3272 | 67.7 | 3805 | 78.7 |
| 4K- | strid.exe | 4837 | 3185 | 65.8 | 3272 | 67.7 | 3805 | 78.7 |
| 6K | shell.com | 3894 | 1072 | 27.5 | 1105 | 28.4 | 1263 | 32.4 |
| | grep2.exe | 5934 | 3667 | 61.8 | 3759 | 63.3 | 4570 | 77.0 |
| | touch.com | 5118 | 3347 | 65.4 | 3431 | 67.0 | 4001 | 78.2 |
| | Avg | 4924 | 2985 | 60.2 | 2747 | 55.8 | 3489 | 70.9 |
| 8K | stup.exe | 7520 | 5402 | 71.8 | 5560 | 73.9 | 6955 | 92.5 |
| 6400- | wpinf.exe | 8192 | 6857 | 83.7 | 5332 | 65.1 | 6617 | 80.8 |
| 9600 | patch.exe | 6788 | 4581 | 67.7 | 4689 | 69.3 | 5819 | 85.7 |
| | grep.com | 7029 | 4599 | 65.4 | 4711 | 67.0 | 5709 | 81.2 |
| | tasm2.exe | 6984 | 4064 | 58.2 | 4194 | 60.1 | 5225 | 74.8 |
| | Avg | 7303 | 5101 | 69.8 | 4897 | 67.1 | 6065 | 83.0 |
| 13K | grab.com | 15842 | 7909 | 49.9 | 8380 | 52.9 | 14479 | 91.4 |
| 10.4- | mips.com | 13312 | 5431 | 40.8 | 5962 | 44.8 | 7472 | 56.1 |
| 15.6 | check.exe | 10043 | 6588 | 65.6 | 6811 | 67.8 | 8159 | 81.2 |
| | share.exe | 13424 | 7508 | 55.9 | 7823 | 58.3 | 9277 | 69.1 |

** For various sizes of Executable Files, Compressed only

| File Size | | Execute | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| | crc.exe | 10659 | 7560 | 70.9 | 7790 | 73.1 | 9463 | 88.8 |
| | Avg | 12656 | 6999 | 55.3 | 7353 | 58.1 | 9770 | 77.2 |
| 20K | pkuz.exe | 23528 | 18125 | 77.0 | 18829 | 80.0 | 24075 | 102. |
| 16K- | reduc.exe | 16505 | 11153 | 67.6 | 11610 | 70.3 | 15326 | 92.9 |
| 24K | st.exe | 17184 | 11136 | 64.8 | 11596 | 67.5 | 16898 | 98.3 |
| | red.exe | 16701 | 11246 | 67.8 | 11700 | 70.1 | 15514 | 92.9 |
| | mcstr.exe | 16395 | 8645 | 52.7 | 9111 | 55.6 | 11244 | 68.6 |
| | Avg | 18063 | 12061 | 66.8 | 12569 | 69.6 | 16611 | 92.0 |
| 40K | pkz.exe | 34296 | 25552 | 74.5 | 26604 | 74.6 | 34170 | 99.6 |
| 32K- | lha.exe | 34283 | 25096 | 73.2 | 26111 | 76.2 | 32687 | 95.3 |
| 48K | dcm.exe | 45212 | 22860 | 50.6 | 24762 | 54.8 | 31658 | 70.0 |
| | copy.exe | 42398 | 19889 | 46.9 | 21428 | 50.5 | 28341 | 66.8 |
| | cfig3.exe | 41984 | 24071 | 57.3 | 25429 | 60.6 | 31974 | 76.2 |
| | Avg | 39635 | 23494 | 59.3 | 24867 | 62.7 | 31766 | 80.1 |
| 70K | chkfd.exe | 59208 | 34620 | 58.5 | 36793 | 62.1 | 47949 | 81.0 |
| 56K- | hdini.exe | 75915 | 41549 | 54.7 | 44976 | 59.2 | 54061 | 71.1 |
| 84K | drc.exe | 84322 | 42471 | 50.4 | 46244 | 54.8 | 58349 | 69.2 |
| | globa.exe | 83854 | 40895 | 48.8 | 44796 | 53.4 | 56723 | 67.6 |
| | local.exe | 58742 | 27988 | 47.6 | 30722 | 52.3 | 42366 | 72.1 |
| | Avg | 72408 | 37505 | 51.8 | 40706 | 56.2 | 51890 | 71.7 |
| 120K | conve.exe | 105141 | 59625 | 56.7 | 64078 | 60.9 | 87955 | 83.7 |
| 96K- | graph.exe | 107520 | 70670 | 65.7 | 74884 | 69.6 | 102987 | 95.8 |
| 144K | ll3.exe | 93399 | 49929 | 53.5 | 53340 | 57.1 | 67811 | 72.6 |
| | ift.exe | 111894 | 22043 | 19.7 | 24919 | 22.3 | 28198 | 25.2 |
| | Avg | 104489 | 50567 | 48.4 | 54305 | 52.0 | 71738 | 68.7 |
| 190K | desig.exe | 175292 | 80785 | 46.1 | 87466 | 49.9 | 140659 | 80.2 |
| 152K- | dash.exe | 197274 | 98509 | 49.9 | 105393 | 53.4 | 151549 | 76.8 |
| 228K | disp.exe | 179560 | 63846 | 35.6 | 70165 | 39.1 | 96975 | 54.0 |
| | dsl.exe | 167734 | 61394 | 36.6 | 67199 | 40.1 | 91795 | 54.7 |
| | exec.exe | 172388 | 65147 | 37.8 | 70683 | 41.0 | 100567 | 58.3 |
| | Avg | 178450 | 73936 | 41.4 | 80181 | 44.9 | 116309 | 65.2 |
| 300K | mcad.exe | 289664 | 142159 | 49.1 | 153675 | 53.1 | 224254 | 77.4 |
| 240K- | check.exe | 351232 | 170400 | 48.5 | 183369 | 52.2 | 253029 | 72.0 |
| 360K | cproc.exe | 376486 | 159581 | 42.4 | 176076 | 46.8 | 257589 | 68.4 |
| | wcsim.exe | 284184 | 107365 | 37.8 | 118984 | 41.9 | 181342 | 63.8 |
| | pcgpp.exe | 273432 | 125300 | 45.8 | 137224 | 50.2 | 195168 | 71.4 |
| | Avg | 315000 | 140961 | 44.7 | 153866 | 48.8 | 222276 | 70.6 |
| 500K | mat38.exe | 428768 | 209473 | 48.9 | 224145 | 52.3 | 303367 | 70.8 |
| 400K- | gpp38.exe | 418612 | 188883 | 45.1 | 204706 | 48.9 | 284042 | 67.9 |
| 600K | stmed.exe | 548640 | 262525 | 47.9 | 284796 | 51.9 | 408439 | 74.4 |

** For various sizes of Executable Files, Compressed only

| File Size | | Execute | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| | probe.exe | 543952 | 244485 | 44.9 | 266796 | 49.0 | 385640 | 70.9 |
| | Avg | 484993 | 226342 | 46.7 | 245111 | 50.5 | 345372 | 71.2 |
| 800K | pshel.exe | 635552 | 301331 | 47.4 | 326990 | 51.4 | 454701 | 71.5 |
| 640K-960K | pspic.exe | 781504 | 352518 | 45.1 | 386205 | 49.4 | 550702 | 70.5 |
| | tc.exe | 887104 | 467688 | 52.7 | 506010 | 57.0 | 645948 | 72.8 |
| | graft.exe | 644029 | 644029 | 100. | 686691 | 107. | 907302 | 141. |
| | Avg | 737047 | 441392 | 59.9 | 476474 | 64.6 | 639663 | 86.8 |
| | Total | 8,576,430 | 4,405,559 | | 4,757,878 | | 6,537,717 | |
| | Ratio | 100 % | 51.4 % | | 55.5 % | | 76.2 % | |

Table 3 COMPRESSED, dBASE Output Files < Fig. 9 >

** For various sizes of dBASE Output Files, Compressed Only

| File Size | | dBASE | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | stokn.dbf | 640 | 382 | 59.7 | 367 | 57.5 | 381 | 59.5 |
| 400- | sysid.dbf | 418 | 190 | 45.5 | 175 | 42.0 | 204 | 48.8 |
| 600 | systi.dbf | 427 | 166 | 38.9 | 149 | 35.0 | 214 | 50.1 |
| | trans.dbf | 640 | 301 | 47.0 | 289 | 45.2 | 333 | 52.0 |
| | Avg | 531 | 260 | 49.0 | 244 | 46.0 | 283 | 58.3 |
| 1K | sales.dbf | 894 | 342 | 38.3 | 342 | 38.3 | 375 | 41.9 |
| 800- | stokp.dbf | 896 | 436 | 48.7 | 423 | 47.3 | 478 | 53.3 |
| 1200 | acctr.dbf | 1280 | 445 | 34.8 | 463 | 36.2 | 551 | 43.0 |
| | codes.dbf | 1152 | 532 | 46.2 | 557 | 48.4 | 549 | 47.7 |
| | items.dbf | 893 | 346 | 38.7 | 352 | 39.4 | 394 | 44.1 |
| | Avg | 1023 | 420 | 41.1 | 427 | 41.1 | 469 | 45.8 |
| 1.8K | clien.dbf | 1664 | 849 | 51.0 | 878 | 52.8 | 881 | 52.9 |
| 1440- | cust.dbf | 2048 | 875 | 42.7 | 910 | 44.5 | 991 | 48.4 |
| 2160 | peopl.dbf | 2048 | 984 | 48.0 | 1033 | 50.4 | 985 | 48.1 |
| | systa.dbf | 1539 | 449 | 29.2 | 478 | 31.1 | 574 | 37.3 |
| | stock.dbf | 1664 | 585 | 35.2 | 602 | 36.2 | 804 | 48.3 |
| | Avg | 1793 | 748 | 41.8 | 780 | 43.5 | 847 | 47.2 |
| 3K | conte.dbf | 2304 | 934 | 40.5 | 970 | 42.1 | 1095 | 47.5 |
| 2400- | custo.dbf | 2666 | 1356 | 50.9 | 1412 | 53.0 | 1501 | 56.3 |
| 3600 | inven.dbf | 2371 | 823 | 34.7 | 853 | 36.0 | 1125 | 47.4 |
| | hal3k.dbf | 3268 | 1483 | 45.4 | 1575 | 48.2 | 1521 | 46.5 |
| | 3k_1.dbf | 3202 | 997 | 31.1 | 1067 | 33.3 | 1217 | 38.0 |
| | Avg | 2762 | 1119 | 40.5 | 1175 | 42.5 | 1292 | 46.8 |
| 5K | goood.dbf | 5120 | 1144 | 22.3 | 1350 | 26.4 | 1653 | 32.3 |
| 4K- | names.dbf | 4096 | 2163 | 52.8 | 2281 | 55.7 | 2258 | 55.1 |
| 6K | sysco.dbf | 5586 | 959 | 17.2 | 1105 | 19.8 | 1506 | 27.0 |
| | dba4.dbf | 4969 | 1552 | 31.2 | 1691 | 34.0 | 2217 | 44.6 |
| | ha5k.dbf | 5296 | 2207 | 41.7 | 2475 | 46.7 | 2298 | 43.4 |
| | Avg | 5013 | 1605 | 31.9 | 1780 | 35.5 | 1986 | 39.6 |
| 8K | syscl.dbf | 7831 | 1447 | 18.5 | 1539 | 20.3 | 2129 | 27.2 |
| 6400- | dba2.dbf | 7842 | 2257 | 28.9 | 2706 | 34.5 | 3317 | 40.5 |
| 9600 | 8k_1.dbf | 8194 | 1924 | 23.5 | 2296 | 28.0 | 2558 | 31.2 |
| | hal8k.dbf | 8260 | 3283 | 39.7 | 3760 | 45.5 | 3476 | 42.1 |
| | 8k_2.dbf | 8194 | 1888 | 23.0 | 2265 | 27.7 | 2551 | 31.1 |
| | Avg | 8064 | 2160 | 26.8 | 2513 | 31.2 | 2806 | 34.8 |
| 13K | emplo.dbf | 12288 | 3615 | 29.4 | 4412 | 35.9 | 4376 | 35.6 |
| 10.4- | dba1.dbf | 12639 | 3339 | 26.4 | 4244 | 33.6 | 3734 | 29.5 |
| 15.6 | offic.dbf | 11261 | 3808 | 33.8 | 4381 | 38.9 | 4831 | 42.9 |
| | h13k.dbf | 13252 | 4873 | 36.8 | 5947 | 44.9 | 5197 | 39.2 |

** For various sizes of dBASE Output Files, Compressed Only

| File Size | dBASE | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|
| qual.dbf | 13453 | 1999 | 14.9 | 2855 | 21.2 | 3107 | 23.1 |
| Avg | 12579 | 3527 | 28.0 | 4368 | 34.7 | 4249 | 33.8 |
| 20K dba3.dbf | 19245 | 4708 | 24.5 | 5539 | 28.8 | 4648 | 24.2 |
| 16K- ofil1.dbf | 16274 | 4259 | 26.2 | 5663 | 34.8 | 5827 | 35.8 |
| ofil2.dbf | 20102 | 4692 | 23.3 | 6736 | 33.5 | 7036 | 35.0 |
| 20k_2.dbf | 20258 | 3948 | 19.5 | 5354 | 26.4 | 5189 | 25.6 |
| h20k.dbf | 20272 | 7181 | 35.4 | 9173 | 45.3 | 7605 | 37.5 |
| Avg | 19230 | 4958 | 25.8 | 6493 | 33.8 | 5260 | 27.4 |
| 40K h40k.dbf | 40240 | 13681 | 34.0 | 17948 | 44.6 | 13714 | 34.1 |
| 32K- ha40k.dbf | 40240 | 13662 | 34.0 | 17910 | 44.5 | 13728 | 34.1 |
| 48K 40k_2.dbf | 40354 | 7389 | 18.3 | 10368 | 25.7 | 9263 | 23.0 |
| 40k_1.dbf | 40354 | 7423 | 18.4 | 10401 | 25.8 | 9191 | 22.8 |
| Avg | 40297 | 10539 | 26.2 | 14157 | 35.1 | 11474 | 28.5 |
| 70K h70k.dbf | 70192 | 23419 | 33.4 | 30876 | 44.0 | 22578 | 32.2 |
| 56K- ha70k.dbf | 70192 | 23333 | 33.2 | 31129 | 44.3 | 22557 | 32.1 |
| 84K 70k_2.dbf | 70530 | 12458 | 17.7 | 17934 | 25.4 | 14695 | 20.8 |
| 70k_1.dbf | 70530 | 12602 | 17.9 | 17897 | 25.4 | 14624 | 20.7 |
| Avg | 70361 | 17953 | 25.5 | 24459 | 34.8 | 18614 | 26.4 |
| 120K h120.dbf | 120190 | 39630 | 33.0 | 52920 | 44.0 | 37239 | 31.0 |
| 96K- ha120.dbf | 120190 | 39676 | 33.0 | 52929 | 44.0 | 37416 | 31.1 |
| 144K 120k1.dbf | 120802 | 21242 | 17.6 | 30609 | 25.3 | 23551 | 19.5 |
| 120k2.dbf | 120802 | 21045 | 17.4 | 30541 | 25.3 | 23637 | 19.6 |
| Avg | 120496 | 30398 | 25.2 | 41750 | 34.6 | 30461 | 25.3 |
| 190K h190.dbf | 190156 | 62354 | 32.8 | 83892 | 44.1 | 57917 | 30.5 |
| 152K-190k2.dbf | 191170 | 33181 | 17.4 | 48186 | 25.2 | 35643 | 18.6 |
| 228K 190k1.dbf | 191170 | 33111 | 17.3 | 48034 | 25.1 | 35882 | 18.8 |
| Avg | 190832 | 42882 | 22.5 | 60037 | 31.5 | 43147 | 22.6 |
| 300K 300k2.dbf | 301762 | 52181 | 17.3 | 76196 | 25.3 | 55825 | 18.5 |
| 240K-300k1.dbf | 301762 | 52090 | 17.3 | 76120 | 25.2 | 55370 | 18.3 |
| 360K Avg | 301762 | 52136 | 17.3 | 76158 | 25.2 | 55598 | 18.4 |
| 500K 500k2.dbf | 502818 | 86410 | 17.2 | 126604 | 25.2 | 93314 | 18.6 |
| 400K-500k1.dbf | 502818 | 86479 | 17.2 | 126627 | 25.2 | 93588 | 18.6 |
| 600K Avg | 502818 | 86445 | 17.2 | 126616 | 25.2 | 93451 | 18.6 |
| 800K zipco.dbf | 967384 | 304450 | 31.5 | 345132 | 35.8 | 360322 | 37.2 |
| 640K-800k2.dbf | 804450 | 138040 | 17.2 | 202127 | 25.1 | 150638 | 18.7 |
| 960K 800k1.dbf | 804450 | 138066 | 17.2 | 202331 | 25.2 | 149402 | 18.6 |
| Avg | 858761 | 193519 | 22.5 | 249863 | 29.1 | 220121 | 25.6 |
| Total | 5,937,002 | 1,295,643 | | 1,745,378 | | 1.419,750 | |
| Ratio | 100 % | 21.8 % | | 29.4 % | | 23.9 % | |

Table 4 COMPRESSED, IMAGE FILES < Fig. 11 >

** For various sizes of Image(Graphic) Files, Compressed

| File Size | Image | | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | augus.svg | 494 | 136 | 27.5 | 124 | 25.2 | 142 | 28.7 |
| 400- | aushh.svg | 494 | 129 | 26.1 | 117 | 23.9 | 136 | 27.5 |
| | pebbl.svg | 494 | 153 | 31.0 | 139 | 28.2 | 155 | 31.4 |
| | grchk.f | 599 | 389 | 64.9 | 370 | 61.8 | 406 | 67.8 |
| | compa | 460 | 294 | 63.9 | 268 | 58.4 | 306 | 66.5 |
| | Avg | 508 | 220 | 43.3 | 204 | 40.2 | 229 | 45.1 |
| 1K | free1.wpg | 1210 | 642 | 53.1 | 638 | 52.8 | 781 | 64.5 |
| 800- | mktbl | 1044 | 732 | 70.1 | 708 | 67.8 | 788 | 75.5 |
| | grlgt.f | 877 | 546 | 62.3 | 527 | 60.1 | 584 | 66.6 |
| | pgcp .f | 893 | 509 | 57.0 | 489 | 54.8 | 573 | 64.2 |
| | pglab.f | 924 | 465 | 50.3 | 451 | 48.9 | 566 | 61.3 |
| | Avg | 990 | 579 | 58.5 | 563 | 56.9 | 658 | 66.5 |
| 1.8K | free3.wpg | 1422 | 691 | 48.6 | 693 | 48.7 | 876 | 61.6 |
| 1440- | fhvst.wpg | 1916 | 1050 | 54.8 | 1067 | 55.7 | 1299 | 67.8 |
| 2160 | free5.wpg | 1644 | 718 | 43.7 | 726 | 44.2 | 985 | 59.9 |
| | free6.wpg | 1618 | 762 | 47.1 | 765 | 47.3 | 995 | 61.5 |
| | Avg | 1650 | 805 | 48.8 | 813 | 49.3 | 1039 | 63.0 |
| 3K | snow.rf | 3478 | 1400 | 40.3 | 1501 | 43.2 | 1729 | 49.7 |
| 2400- | patti.shp | 2432 | 745 | 30.6 | 789 | 32.5 | 1179 | 48.5 |
| 3600 | headc. | 2842 | 1459 | 51.3 | 1491 | 52.5 | 1783 | 62.7 |
| | metal | 2536 | 1508 | 59.5 | 1529 | 60.3 | 1721 | 67.9 |
| | fjamm.wpg | 2412 | 1159 | 48.1 | 1189 | 49.3 | 1475 | 61.2 |
| | grap2.wpg | 3558 | 1178 | 33.1 | 1200 | 33.7 | 1729 | 48.6 |
| | Avg | 2876 | 1242 | 43.2 | 1283 | 44.6 | 1603 | 55.7 |
| 5K | verti.vrs | 4945 | 1449 | 29.3 | 1577 | 31.9 | 2527 | 51.1 |
| 4K- | fonti.shp | 4096 | 1855 | 45.3 | 1957 | 47.8 | 2157 | 52.7 |
| 6K | haa1.dwg | 4368 | 1518 | 34.8 | 1876 | 42.9 | 2020 | 46.5 |
| | colo | 5860 | 2591 | 44.2 | 2928 | 50.0 | 3128 | 53.4 |
| | garfi.im1 | 4961 | 2493 | 50.3 | 2639 | 53.2 | 2723 | 54.9 |
| | grope.f | 4538 | 1853 | 40.8 | 1968 | 43.4 | 2342 | 51.6 |
| | Avg | 4795 | 1960 | 40.9 | 2158 | 45.0 | 2483 | 51.8 |
| 8K | e3830.dwg | 8464 | 2596 | 30.7 | 3099 | 36.6 | 3556 | 42.0 |
| 6400- | etbl | 8379 | 3379 | 40.3 | 3855 | 46.0 | 4492 | 53.6 |
| 9600 | imdri.f | 8942 | 2835 | 31.7 | 3367 | 37.7 | 3875 | 43.3 |
| | sunvi.sha | 7685 | 2978 | 38.8 | 3406 | 44.3 | 4063 | 52.9 |
| | syn.me | 9147 | 2660 | 29.1 | 3242 | 35.4 | 3553 | 38.8 |
| | teapo | 8227 | 2887 | 35.1 | 3337 | 40.6 | 4028 | 49.0 |
| | Avg | 8474 | 2889 | 34.1 | 3386 | 40.0 | 3928 | 46.4 |
| 13K | arch.dat | 13717 | 2912 | 21.2 | 3225 | 23.5 | 3877 | 28.3 |

** For various sizes of Image(Graphic) Files, Compressed

| File Size | | Image | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|---|
| 10.4- | bearl.rf | 15200 | 3888 | 25.6 | 4594 | 30.2 | 3773 | 24.8 |
| 15.6 | geniu.vrs | 12361 | 5112 | 41.4 | 5337 | 43.2 | 6140 | 49.7 |
| | main.shp | 11264 | 4915 | 43.6 | 5242 | 46.5 | 6182 | 54.9 |
| | thes2.dwg | 11984 | 4620 | 38.6 | 5518 | 46.0 | 5579 | 46.6 |
| | Avg | 12905 | 4289 | 33.2 | 4783 | 37.0 | 5110 | 39.6 |
| 20K | clown.rf | 17816 | 6156 | 34.6 | 6866 | 38.5 | 6133 | 43.4 |
| 16K- | turk.rf | 19288 | 9219 | 47.8 | 10013 | 51.9 | 9803 | 50.8 |
| 24K | aero.eps | 21577 | 6723 | 31.2 | 7630 | 35.4 | 9013 | 41.8 |
| | bord.shp | 20608 | 7525 | 36.5 | 8054 | 39.1 | 10574 | 51.3 |
| | tsai.dwg | 18688 | 8280 | 44.3 | 9550 | 51.2 | 10355 | 55.4 |
| | Avg | 19595 | 7581 | 38.7 | 8423 | 42.9 | 9176 | 46.8 |
| 40K | golf.dat | 50186 | 7102 | 14.2 | 8700 | 17.3 | 11871 | 23.7 |
| 32K- | birds.rf | 47865 | 17377 | 36.3 | 19056 | 39.8 | 19189 | 40.1 |
| 48K | scree.rf | 38147 | 3124 | 8.2 | 4027 | 10.6 | 4309 | 11.3 |
| | sql.sha | 44976 | 12175 | 27.1 | 15942 | 35.4 | 20129 | 44.8 |
| | img8.rgb | 30752 | 4861 | 15.8 | 5220 | 17.0 | 6785 | 22.1 |
| | Avg | 42385 | 8928 | 21.1 | 10589 | 25.0 | 12457 | 29.4 |
| 70K | slib.shp | 70400 | 40863 | 58.0 | 43521 | 61.8 | 49120 | 69.8 |
| 56K- | bv.sr | 84432 | 68807 | 81.5 | 72408 | 85.8 | 74273 | 88.0 |
| | bfg.sr | 77089 | 64653 | 83.9 | 68568 | 88.9 | 68073 | 88.3 |
| | show | 82177 | 30978 | 37.7 | 33273 | 40.5 | 50642 | 61.6 |
| | xhip | 82174 | 31758 | 38.6 | 34061 | 41.4 | 45660 | 55.6 |
| | Avg | 79254 | 47412 | 59.8 | 50366 | 63.6 | 57554 | 72.6 |
| 120K | augus.m18 | 111864 | 27412 | 24.5 | 32016 | 28.6 | 30434 | 27.2 |
| 96K- | bush.m18 | 111864 | 24910 | 22.3 | 29759 | 26.6 | 28922 | 25.9 |
| 144K | peb.m18 | 111864 | 24980 | 22.3 | 29282 | 26.2 | 28804 | 25.7 |
| | lenno.im1 | 129632 | 36770 | 28.3 | 42213 | 32.6 | 31095 | 24.0 |
| | movie | 98563 | 38709 | 39.3 | 41754 | 42.4 | 57196 | 58.0 |
| | space.im1 | 129700 | 22373 | 17.2 | 27509 | 21.2 | 20509 | 15.8 |
| | Avg | 115581 | 29192 | 25.3 | 33756 | 29.2 | 32827 | 28.4 |
| 190K | plant.mif | 177980 | 30302 | 17.0 | 44557 | 25.0 | 45103 | 25.3 |
| 152K- | bdy2.cbd | 228799 | 114240 | 49.9 | 118052 | 51.6 | 107813 | 47.1 |
| 228K | img5.rgb | 223800 | 181291 | 81.0 | 191694 | 85.7 | 215044 | 96.1 |
| | img9.rgb | 200427 | 139112 | 69.4 | 148318 | 74.0 | 136806 | 68.3 |
| | img14.eps | 176370 | 73887 | 41.0 | 90457 | 51.3 | 74179 | 42.1 |
| | Avg | 201475 | 107766 | 53.5 | 118616 | 58.9 | 115789 | 57.5 |
| 300K | ad.eps | 320174 | 108499 | 33.9 | 127433 | 39.8 | 131753 | 41.2 |
| 240K- | img13.rle | 243696 | 149260 | 61.2 | 162506 | 66.7 | 139605 | 57.3 |
| 360K | bdy.cbd | 261981 | 134222 | 51.2 | 141145 | 53.9 | 178550 | 68.2 |
| | libpg.a | 362473 | 147958 | 40.8 | 170598 | 47.1 | 180430 | 49.8 |

** For various sizes of Image(Graphic) Files, Compressed and Archived

| File Size | Image | PKZIP | | StacPack | | Compress | |
|---|---|---|---|---|---|---|---|
| Avg | 297081 | 134985 | 45.4 | 150424 | 50.6 | 157585 | 53.0 |
| 500K 944gt.scr | 494267 | 207812 | 42.0 | 226309 | 45.8 | 294227 | 59.5 |
| 400K-bab02.eps | 526772 | 171576 | 32.6 | 217306 | 41.3 | 169695 | 32.2 |
| 600K 63vet.scr | 471937 | 171599 | 36.4 | 187271 | 39.7 | 272544 | 57.8 |
| b2.scr | 424532 | 303021 | 71.4 | 321548 | 75.7 | 291383 | 68.6 |
| Avg | 479377 | 213502 | 44.5 | 238109 | 49.7 | 256962 | 53.6 |
| 800K bigk.scr | 767399 | 247580 | 32.3 | 273665 | 35.7 | 430254 | 56.1 |
| 640K- ball.scr | 742684 | 386224 | 52.0 | 419076 | 56.4 | 471329 | 63.5 |
| 960K beac.scr | 803894 | 540307 | 67.2 | 571313 | 71.1 | 534562 | 66.5 |
| half.scr | 961208 | 662157 | 68.9 | 708721 | 73.7 | 646509 | 67.3 |
| solin.sc | 1070111 | 820446 | 76.7 | 884196 | 82.6 | 833651 | 77.9 |
| Avg | 869059 | 531343 | 61.1 | 571406 | 65.7 | 583261 | 67.1 |
| Total | 10,033,651 | 5,149,569 | | 5,625,605 | | 5,828,549 | |
| Ratio | 100 % | 51.3 % | | 56.1 % | | 58.1 % | |

## APPENDIX B. RESULT OF EXPERIMENT FOR ARCHIVING SOFTWARE

Table 5 COMPRESSED AND ARCHIVED, TEXT FILES < Fig. 6 >

** For various Sizes of Text Files, Compressed and Archived

| File Size | | Text | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | shutt.mcd | 555 | 345 | 62.2 | 345 | 62.2 | 412 | 74.2 |
| 400- | oilri.mcd | 554 | 345 | 62.3 | 344 | 62.1 | 411 | 74.2 |
| 600 | spira.mcd | 640 | 407 | 63.6 | 407 | 63.6 | 486 | 75.9 |
| | cond.m | 446 | 291 | 65.2 | 291 | 65.2 | 343 | 76.9 |
| | dec2h.m | 555 | 347 | 62.5 | 347 | 62.5 | 429 | 77.3 |
| | Avg | 550 | 347 | 63.1 | 347 | 63.1 | 416 | 75.6 |
| 1K | feath.m | 1207 | 661 | 54.8 | 661 | 54.8 | 845 | 70.0 |
| 800- | anhar.mcd | 1025 | 623 | 60.8 | 623 | 60.8 | 757 | 73.9 |
| 1.2K | polar.mcd | 809 | 502 | 62.1 | 501 | 61.9 | 603 | 74.5 |
| | hex2n.m | 1053 | 580 | 55.1 | 580 | 55.1 | 729 | 69.2 |
| | expm1.m | 804 | 467 | 58.1 | 467 | 58.1 | 574 | 71.4 |
| | Avg | 980 | 567 | 57.9 | 566 | 57.8 | 702 | 71.6 |
| 1.8K | bode.mcd | 2258 | 1218 | 53.9 | 1218 | 53.9 | 1427 | 63.2 |
| 1440- | boole.mcd | 1455 | 761 | 52.3 | 761 | 52.3 | 990 | 68.0 |
| 2160 | brake.mcd | 1947 | 1000 | 51.4 | 1000 | 51.4 | 1203 | 61.8 |
| | compf.mcd | 1528 | 806 | 52.7 | 806 | 52.7 | 964 | 63.1 |
| | erf.m | 2062 | 1010 | 49.0 | 1010 | 49.0 | 1168 | 56.6 |
| | Avg | 1850 | 959 | 51.8 | 959 | 51.8 | 1150 | 62.2 |
| 3K | anten.doc | 2737 | 1370 | 50.1 | 1371 | 50.1 | 1504 | 55.0 |
| 2400- | mks.mcd | 3772 | 1671 | 44.3 | 1671 | 44.3 | 1877 | 49.8 |
| 3600 | besse.m | 2426 | 1165 | 48.0 | 1165 | 48.0 | 1336 | 55.1 |
| | bilin.m | 3076 | 1400 | 45.5 | 1401 | 45.5 | 1603 | 52.1 |
| | cplxp.m | 3021 | 1317 | 43.6 | 1317 | 43.6 | 1462 | 48.4 |
| | Avg | 3006 | 1385 | 46.1 | 1385 | 46.1 | 1556 | 51.8 |
| 5K | read1.doc | 4259 | 2034 | 47.8 | 2035 | 47.8 | 2247 | 52.8 |
| 4K- | inst.doc | 4029 | 1788 | 44.4 | 1788 | 44.4 | 1948 | 48.3 |
| | readm.txt | 5594 | 2258 | 40.4 | 2260 | 40.4 | 2491 | 44.5 |
| | cgs.mcd | 4383 | 1900 | 44.3 | 1902 | 43.4 | 2080 | 47.5 |
| | direc.mcd | 5112 | 2066 | 40.4 | 2067 | 40.4 | 2291 | 44.8 |
| | Avg | 4675 | 2009 | 43.0 | 2010 | 43.0 | 2211 | 47.3 |
| 8K | stmed.msg | 7900 | 2892 | 36.6 | 2894 | 36.6 | 3226 | 40.8 |
| 6400- | redm.mcd | 7615 | 3421 | 44.9 | 3422 | 44.9 | 3659 | 48.0 |
| 9600 | bench.m | 7377 | 2436 | 33.0 | 2437 | 33.0 | 2746 | 37.2 |
| | spi2.dat | 9449 | 1832 | 19.4 | 1831 | 19.4 | 2264 | 24.0 |
| | fload.c | 8727 | 2699 | 30.9 | 2699 | 30.9 | 3017 | 34.6 |

** For various sizes of Text Files, Compressed and Archived

| File Size | | Text | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| | Avg | 8213 | 2656 | 32.3 | 2657 | 32.3 | 2982 | 36.3 |
| 13K | api.doc | 15240 | 5659 | 37.1 | 5667 | 37.2 | 6132 | 40.2 |
| 10.4- | textb.doc | 15429 | 6712 | 43.5 | 6693 | 43.4 | 7662 | 49.7 |
| 15.6 | read.doc | 15443 | 5627 | 36.4 | 5655 | 36.6 | 5980 | 38.7 |
| | remez.m | 15407 | 4291 | 27.0 | 4289 | 27.8 | 4765 | 30.9 |
| | read3.doc | 12006 | 4748 | 39.5 | 4752 | 39.6 | 5021 | 41.8 |
| | Avg | 14705 | 5407 | 45.0 | 5411 | 38.4 | 5912 | 42.0 |
| 20K | thesi.doc | 17408 | 5936 | 34.1 | 5938 | 34.1 | 6513 | 38.2 |
| 16K- | arrow.doc | 21582 | 9978 | 46.2 | 9940 | 46.1 | 11044 | 51.2 |
| 24K | cshel.doc | 24911 | 7499 | 30.1 | 7605 | 30.5 | 8061 | 32.4 |
| | redu.c | 21931 | 4674 | 21.3 | 4708 | 21.5 | 5210 | 23.8 |
| | spi1.dat | 21477 | 6186 | 28.8 | 6119 | 28.5 | 6820 | 31.8 |
| | Avg | 21461 | 6855 | 31.9 | 6862 | 32.0 | 7530 | 35.1 |
| 40K | chara.doc | 42223 | 12922 | 30.6 | 13132 | 31.1 | 14167 | 33.6 |
| 32K- | matla.hlp | 50425 | 19446 | 38.6 | 20289 | 40.2 | 21239 | 42.1 |
| 48K | setup.inf | 50014 | 12415 | 24.8 | 12551 | 25.1 | 13479 | 27.0 |
| | eval.lib | 52515 | 15011 | 28.6 | 15327 | 29.2 | 16291 | 31.0 |
| | parts.hlp | 33583 | 8846 | 26.3 | 9266 | 27.6 | 9899 | 29.5 |
| | Avg | 45752 | 13728 | 30.0 | 14113 | 30.8 | 15015 | 32.8 |
| 70K | holid.doc | 55584 | 30664 | 55.2 | 30556 | 55.0 | 32194 | 57.9 |
| 56K- | mcad.hlp | 53184 | 13171 | 24.8 | 13291 | 25.0 | 14860 | 27.9 |
| 84K | check.hlp | 52616 | 16849 | 32.0 | 17441 | 33.1 | 18217 | 34.6 |
| | util.doc | 79144 | 23823 | 30.1 | 24488 | 30.9 | 25533 | 32.3 |
| | class.doc | 55736 | 13403 | 24.0 | 13793 | 24.7 | 14811 | 26.6 |
| | Avg | 59253 | 19582 | 33.0 | 19914 | 33.6 | 21123 | 35.6 |
| 120K | qbasi.hlp | 130810 | 104803 | 80.1 | 107152 | 81.9 | 108785 | 83.2 |
| 96K- | anlg.lib | 138727 | 17247 | 12.4 | 17484 | 12.6 | 21367 | 15.4 |
| 144K | tex.lib | 131653 | 8477 | 6.4 | 8787 | 6.7 | 12176 | 9.2 |
| | thyri.lib | 135346 | 8334 | 6.2 | 8352 | 6.2 | 11172 | 8.3 |
| | lin.lib | 110682 | 12018 | 10.9 | 13232 | 12.0 | 15931 | 14.4 |
| | Avg | 129444 | 30176 | 23.3 | 31001 | 23.9 | 33886 | 26.2 |
| 190K | ssims.mdr | 212493 | 18510 | 8.7 | 17958 | 8.5 | 22449 | 10.6 |
| 152K- | evals.dat | 159201 | 91891 | 57.7 | 92303 | 58.0 | 95799 | 60.2 |
| 228K | bipol.lib | 185420 | 18868 | 10.2 | 21710 | 11.7 | 26120 | 14.1 |
| | diode.lib | 158181 | 16226 | 10.3 | 19024 | 12.0 | 22357 | 14.1 |
| | pwr.lib | 184757 | 17372 | 9.4 | 18174 | 9.8 | 21980 | 11.9 |
| | Avg | 180010 | 32573 | 18.1 | 33942 | 18.9 | 33039 | 18.4 |
| 300K | quatt.hlp | 287589 | 96290 | 33.5 | 100159 | 34.8 | 103045 | 35.8 |
| 240K- | | | | | | | | |
| 360K | Avg | 287589 | 96290 | 33.5 | 100159 | 34.8 | 103045 | 35.8 |

** For various sizes of Text Files, Compressed and Archived

| File Size | | Text | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| 500K | ridm.txt | 454374 | 136477 | 30.0 | 145273 | 32.0 | 151119 | 33.3 |
| 400K-600K | Avg | 454374 | 136477 | 30.0 | 145273 | 32.0 | 151119 | 33.3 |
| 800K | tchel.tch | 976250 | 444057 | 45.5 | 469940 | 48.1 | 477828 | 48.9 |
| 640K-960K | Avg | 976250 | 444057 | 45.5 | 469940 | 48.1 | 477828 | 48.9 |
| Total | | 4,067,716 | 1,258,142 | | 1,310,669 | | 1,383,388 | |
| Ratio | | 100 % | 30.9 % | | 32.2 % | | 34.0 % | |

Table 6 COMPRESSED AND ARCHIVED, EXECUTABLE FILES < Fig.8 >

** For various sizes of Executable Files, Compressed and Archived

| File Size | Executables | | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | isat.exe | 568 | 85 | 15.0 | 85 | 15.0 | 80 | 14.1 |
| 400- | chkri.com | 688 | 570 | 82.8 | 570 | 82.8 | 595 | 86.5 |
| 600 | rambi.com | 307 | 252 | 82.1 | 252 | 82.1 | 268 | 87.3 |
| | exet1.com | 413 | 407 | 98.5 | 406 | 98.3 | 400 | 96.9 |
| | fasto.exe | 680 | 198 | 29.1 | 198 | 29.1 | 193 | 28.4 |
| | Avg | 531 | 302 | 56.9 | 302 | 56.9 | 307 | 57.8 |
| 1K | egaep.com | 1006 | 640 | 63.6 | 639 | 63.5 | 684 | 68.0 |
| 800- | gen41.exe | 1125 | 103 | 9.2 | 103 | 9.2 | 85 | 7.6 |
| 1200 | loadf.com | 1131 | 574 | 50.8 | 574 | 50.8 | 665 | 58.8 |
| | prtsc.exe | 1176 | 418 | 35.5 | 418 | 35.5 | 419 | 35.6 |
| | Avg | 1110 | 434 | 39.1 | 434 | 39.1 | 463 | 41.7 |
| 1.8K | curso.com | 1452 | 1123 | 77.3 | 1123 | 77.3 | 1226 | 84.4 |
| 1440- | gen42.exe | 1477 | 157 | 10.6 | 157 | 10.6 | 166 | 11.2 |
| 2160 | 67ves.com | 1559 | 964 | 61.8 | 965 | 61.9 | 1401 | 89.9 |
| | runti.exe | 1590 | 713 | 44.8 | 714 | 44.9 | 1067 | 67.1 |
| | dbase.exe | 1588 | 714 | 45.0 | 715 | 45.0 | 1065 | 67.1 |
| | Avg | 1533 | 734 | 47.9 | 735 | 47.9 | 985 | 64.3 |
| 3K | egala.com | 2388 | 1100 | 46.1 | 1100 | 46.1 | 1504 | 63.0 |
| 2400- | more.com | 2618 | 1971 | 75.3 | 1971 | 75.3 | 2480 | 94.7 |
| 3600 | appen.exe | 2902 | 2770 | 95.5 | 2769 | 95.4 | 2902 | 100.0 |
| | setna.exe | 3174 | 1916 | 60.4 | 1916 | 60.4 | 2440 | 76.9 |
| | astcl.com | 2557 | 1736 | 67.9 | 1735 | 67.9 | 2232 | 87.3 |
| | Avg | 2728 | 1899 | 69.6 | 1898 | 69.6 | 2312 | 84.8 |
| 5K | edit.exe | 4837 | 3095 | 64.0 | 3095 | 64.0 | 3654 | 75.5 |
| 4K- | strid.exe | 4837 | 3095 | 64.0 | 3095 | 64.0 | 3654 | 75.5 |
| 6K | shell.com | 3894 | 1007 | 25.9 | 1006 | 25.8 | 1483 | 38.1 |
| | grep2.exe | 5934 | 3539 | 59.6 | 3540 | 59.7 | 4143 | 69.8 |
| | touch.com | 5118 | 3248 | 63.5 | 3248 | 63.5 | 3829 | 74.8 |
| | Avg | 4924 | 2797 | 56.8 | 2797 | 56.8 | 3353 | 68.1 |
| 8K | stup.exe | 7520 | 5264 | 70.0 | 5264 | 70.0 | 5889 | 78.3 |
| 6400- | wpinf.exe | 8192 | 5092 | 62.2 | 5093 | 62.2 | 5748 | 70.2 |
| 9600 | patch.exe | 6788 | 4417 | 65.3 | 4417 | 65.3 | 5027 | 74.1 |
| | grep.com | 7029 | 4519 | 64.3 | 4518 | 64.3 | 5168 | 73.5 |
| | tasm2.exe | 6984 | 3933 | 56.3 | 3933 | 56.3 | 4585 | 65.7 |
| | Avg | 7303 | 4645 | 63.6 | 4645 | 63.6 | 5283 | 72.3 |
| 13K | grab.com | 15842 | 7818 | 49.3 | 7820 | 49.4 | 8632 | 54.5 |
| 10.4- | mips.com | 13312 | 5149 | 38.7 | 5150 | 38.7 | 6120 | 46.0 |
| 15.6 | check.exe | 10043 | 6393 | 63.7 | 6391 | 63.6 | 7048 | 70.2 |

** For various sizes of Executable Files, Compressed and Archived

| File Size | Executables | | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| | share.exe | 13424 | 7266 | 54.1 | 7251 | 54.0 | 8036 | 59.9 |
| | crc.exe | 10659 | 7353 | 69.0 | 7349 | 68.9 | 8016 | 75.2 |
| | Avg | 12656 | 6796 | 53.7 | 6792 | 53.7 | 7570 | 59.8 |
| 20K | pkuz.exe | 23528 | 17491 | 74.3 | 17490 | 74.3 | 18638 | 78.1 |
| 16K- | reduc.exe | 16505 | 10916 | 66.1 | 10899 | 66.0 | 11633 | 70.5 |
| | st.exe | 17184 | 10852 | 63.2 | 10840 | 63.1 | 11581 | 67.4 |
| | red.exe | 16701 | 11002 | 65.9 | 10987 | 65.8 | 11727 | 70.2 |
| | mcstr.exe | 16395 | 8419 | 51.4 | 8406 | 51.3 | 9181 | 56.0 |
| | Avg | 18063 | 11736 | 65.0 | 11724 | 64.9 | 12552 | 69.5 |
| 40K | pkz.exe | 34296 | 24487 | 71.4 | 24633 | 71.8 | 25569 | 74.6 |
| 32K- | lha.exe | 34283 | 24477 | 71.4 | 24611 | 71.8 | 25521 | 74.4 |
| | dcm.exe | 45212 | 22004 | 48.7 | 22139 | 49.0 | 23307 | 51.6 |
| | copy.exe | 42398 | 18672 | 44.0 | 18934 | 44.7 | 20214 | 47.7 |
| | cfig3.exe | 41984 | 23392 | 55.7 | 23472 | 55.9 | 24558 | 58.5 |
| | Avg | 39635 | 22606 | 57.0 | 22758 | 57.4 | 23834 | 60.1 |
| 70K | chkfd.exe | 59208 | 33522 | 56.6 | 33711 | 56.9 | 35162 | 59.4 |
| 56K- | hdini.exe | 75915 | 38888 | 51.2 | 39602 | 52.2 | 41444 | 54.6 |
| 84K | drc.exe | 84322 | 40100 | 47.6 | 40587 | 48.1 | 42226 | 50.1 |
| | globa.exe | 83854 | 38833 | 46.3 | 39449 | 47.0 | 40979 | 48.9 |
| | local.exe | 58742 | 26650 | 45.4 | 26989 | 45.9 | 28313 | 48.2 |
| | Avg | 72408 | 35599 | 49.2 | 36068 | 50.1 | 37625 | 52.0 |
| 120K | conve.exe | 105141 | 55826 | 53.1 | 56929 | 54.1 | 59396 | 56.5 |
| 96K- | graph.exe | 107520 | 67494 | 62.8 | 67977 | 63.2 | 69826 | 64.9 |
| 144K | ll3.exe | 93399 | 47234 | 50.6 | 47875 | 51.3 | 50096 | 53.6 |
| | ift.exe | 111894 | 20825 | 18.6 | 20864 | 18.6 | 24003 | 21.5 |
| | Avg | 104489 | 47845 | 45.8 | 48411 | 46.3 | 50838 | 48.7 |
| 190K | desig.exe | 175292 | 72765 | 41.5 | 74001 | 42.2 | 80172 | 41.5 |
| 152K- | dash.exe | 197274 | 93040 | 47.2 | 94191 | 47.7 | 99256 | 50.3 |
| 228K | disp.exe | 179560 | 57256 | 31.9 | 58745 | 32.7 | 63873 | 35.6 |
| | dsl.exe | 167734 | 55602 | 33.1 | 57132 | 34.1 | 62304 | 37.1 |
| | exec.exe | 172388 | 57193 | 33.2 | 58171 | 33.7 | 66092 | 38.3 |
| | Avg | 178450 | 67171 | 37.6 | 68448 | 38.4 | 74339 | 41.7 |
| 300K | mcad.exe | 289664 | 130528 | 45.1 | 132901 | 45.9 | 142049 | 49.0 |
| 240K- | check.exe | 351232 | 155974 | 44.4 | 158586 | 45.2 | 172534 | 49.1 |
| 360K | cproc.exe | 376486 | 141780 | 37.7 | 145546 | 38.7 | 158775 | 42.2 |
| | wcsim.exe | 284184 | 94761 | 33.3 | 97933 | 34.5 | 106577 | 37.5 |
| | pcgpp.exe | 273432 | 110969 | 40.6 | 113517 | 41.5 | 122113 | 44.7 |
| | Avg | 315000 | 126802 | 40.3 | 129697 | 41.2 | 140410 | 44.6 |
| 500K | mat38.exe | 428768 | 199529 | 46.5 | 201689 | 47.0 | 211639 | 49.4 |

** For various sizes of Executable Files, Compressed and Archived

| File Size | Executables | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|
| 400K-gpp38.exe | 418612 | 175359 | 41.9 | 177211 | 42.3 | 188172 | 45.0 |
| 600K stmed.exe | 548640 | 244221 | 44.5 | 248064 | 45.2 | 264243 | 48.2 |
| probe.exe | 543952 | 227871 | 41.9 | 231680 | 42.6 | 245129 | 45.1 |
| Avg | 484993 | 211745 | 43.7 | 214661 | 44.3 | 227296 | 46.9 |
| 800K pshel.exe | 635552 | 276279 | 43.5 | 281149 | 44.2 | 307440 | 48.4 |
| 640K-pspic.exe | 781504 | 324796 | 41.6 | 331450 | 42.4 | 359426 | 46.0 |
| 960K tc.exe | 887104 | 434990 | 49.0 | 442889 | 49.9 | 459652 | 51.8 |
| graft.exe | 644029 | 623980 | 96.9 | 624399 | 97.0 | 626453 | 97.3 |
| Avg | 737047 | 415011 | 56.3 | 419972 | 57.0 | 438243 | 59.5 |
| Total | 8,576,430 | 4,105,576 | | 4,163,228 | | 4,224,910 | |
| Ratio | 100 % | 47.9 % | | 48.5 % | | 49.3 % | |

Table 7 COMPRESSED AND ARCHIVED, dBASE Output Files <Fig.10>

** For various sizes of dBASE Output Files, Compressed and Archived

| File Size | | dBASE | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | stokn.dbf | 640 | 314 | 49.1 | 314 | 49.1 | 373 | 58.3 |
| 400- | sysid.dbf | 418 | 158 | 37.8 | 158 | 37.8 | 194 | 46.4 |
| 600 | systi.dbf | 427 | 142 | 33.3 | 142 | 33.3 | 208 | 48.7 |
| | trans.dbf | 640 | 248 | 38.8 | 248 | 38.8 | 316 | 49.4 |
| | Avg | 531 | 216 | 40.7 | 216 | 40.7 | 273 | 51.4 |
| 1K | sales.dbf | 894 | 279 | 31.2 | 279 | 31.2 | 369 | 41.3 |
| 800- | stokp.dbf | 896 | 370 | 41.3 | 370 | 41.3 | 467 | 52.1 |
| 1200 | acctr.dbf | 1280 | 397 | 31.0 | 397 | 31.0 | 509 | 39.8 |
| | codes.dbf | 1152 | 456 | 39.6 | 455 | 39.5 | 518 | 45.0 |
| | items.dbf | 893 | 300 | 33.6 | 300 | 33.6 | 370 | 41.4 |
| | Avg | 1023 | 340 | 33.2 | 360 | 35.2 | 447 | 43.7 |
| 1.8K | clien.dbf | 1664 | 739 | 44.4 | 735 | 44.2 | 958 | 57.6 |
| 1440- | cust.dbf | 2048 | 770 | 37.6 | 768 | 37.5 | 988 | 48.2 |
| 2160 | peopl.dbf | 2048 | 849 | 41.5 | 845 | 41.3 | 1085 | 53.0 |
| | systa.dbf | 1539 | 384 | 25.0 | 385 | 25.0 | 559 | 36.3 |
| | stock.dbf | 1664 | 499 | 30.0 | 499 | 30.0 | 652 | 39.2 |
| | Avg | 1793 | 648 | 36.1 | 646 | 36.0 | 848 | 47.3 |
| 3K | conte.dbf | 2304 | 826 | 35.9 | 826 | 35.9 | 1055 | 45.8 |
| 2400- | custo.dbf | 2666 | 1195 | 44.8 | 1189 | 44.6 | 1430 | 53.6 |
| 3600 | inven.dbf | 2371 | 702 | 29.6 | 702 | 29.6 | 841 | 35.5 |
| | hal3k.dbf | 3268 | 1287 | 39.4 | 1280 | 39.2 | 1499 | 45.9 |
| | 3k_1.dbf | 3202 | 849 | 26.5 | 836 | 26.1 | 1018 | 31.8 |
| | Avg | 2762 | 972 | 35.2 | 967 | 35.0 | 1169 | 42.3 |
| 5K | goood.dbf | 5120 | 984 | 19.2 | 973 | 19.0 | 1277 | 24.9 |
| 4K- | names.dbf | 4096 | 1938 | 47.3 | 1929 | 47.1 | 2166 | 52.9 |
| 6K | sysco.dbf | 5586 | 808 | 14.5 | 809 | 14.5 | 1602 | 28.7 |
| | dba4.dbf | 4969 | 1377 | 27.7 | 1381 | 27.8 | 1683 | 33.9 |
| | ha5k.dbf | 5296 | 1933 | 36.5 | 1921 | 36.3 | 2165 | 40.9 |
| | Avg | 5034 | 1408 | 28.0 | 1403 | 27.9 | 1779 | 35.3 |
| 8K | syscl.dbf | 7831 | 1171 | 15.0 | 1169 | 14.9 | 1473 | 18.8 |
| 6400- | dba2.dbf | 7842 | 2088 | 26.6 | 2071 | 26.4 | 2445 | 31.2 |
| 9600 | 8k_1.dbf | 8194 | 1601 | 19.5 | 1593 | 19.4 | 1777 | 21.7 |
| | hal8k.dbf | 8260 | 2901 | 35.1 | 2895 | 35.0 | 3172 | 38.4 |
| | 8k_2.dbf | 8194 | 1572 | 19.2 | 1558 | 19.0 | 1736 | 21.2 |
| | Avg | 8064 | 1867 | 23.2 | 1857 | 23.0 | 2121 | 26.3 |
| 13K | emplo.dbf | 12288 | 3403 | 27.7 | 3359 | 27.3 | 3916 | 31.9 |
| 10.4- | dba1.dbf | 12639 | 3158 | 25.0 | 3038 | 24.0 | 3552 | 28.1 |
| 15.6 | offic.dbf | 11261 | 3522 | 31.3 | 3500 | 31.1 | 3977 | 35.3 |

** For various sizes of dBASE Output Files, Compressed and Archived

| File Size | | dBASE | ARJ221A | | LHA213 | | PK251 | |
|---|---|---|---|---|---|---|---|---|
| | h13k.dbf | 13252 | 4409 | 33.3 | 4373 | 33.0 | 4730 | 38.7 |
| | qual.dbf | 13453 | 1659 | 12.3 | 1651 | 12.3 | 1926 | 14.3 |
| | Avg | 12579 | 3230 | 25.7 | 3184 | 25.3 | 3620 | 28.8 |
| 20K | dba3.dbf | 19245 | 4053 | 21.1 | 4036 | 21.0 | 4622 | 24.0 |
| 16K- | ofil1.dbf | 16274 | 3953 | 24.3 | 3908 | 24.0 | 4439 | 27.3 |
| 24k | ofil2.dbf | 20102 | 4339 | 21.6 | 4297 | 21.4 | 4913 | 24.4 |
| | 20k_2.dbf | 20258 | 3233 | 16.0 | 3212 | 15.9 | 3445 | 17.0 |
| | h20k.dbf | 20272 | 6446 | 31.8 | 6439 | 31.8 | 6988 | 34.5 |
| | Avg | 19230 | 4405 | 22.9 | 4378 | 22.8 | 4881 | 25.4 |
| 40K | h40k.dbf | 40240 | 12198 | 30.3 | 12304 | 30.6 | 13182 | 32.8 |
| 32K- | ha40k.dbf | 40240 | 12235 | 30.4 | 12296 | 30.6 | 13181 | 32.8 |
| 48K | 40k_2.dbf | 40354 | 5928 | 14.7 | 5964 | 14.8 | 6252 | 15.5 |
| | 40k_1.dbf | 40354 | 5964 | 14.8 | 6019 | 14.9 | 6352 | 15.7 |
| | Avg | 40297 | 9081 | 22.5 | 9146 | 22.7 | 9742 | 24.2 |
| 70K | h70k.dbf | 70192 | 20647 | 29.4 | 21023 | 30.0 | 22058 | 31.4 |
| 56K- | ha70k.dbf | 70192 | 20625 | 29.4 | 20979 | 29.9 | 22381 | 31.9 |
| 84K | 70k_2.dbf | 70530 | 9795 | 13.9 | 9951 | 14.1 | 10427 | 14.8 |
| | 70k_1.dbf | 70530 | 9943 | 14.1 | 10056 | 14.3 | 10586 | 15.0 |
| | Avg | 70361 | 15253 | 21.7 | 15502 | 22.0 | 16363 | 23.3 |
| 120K | h120.dbf | 120190 | 34692 | 28.9 | 35615 | 29.6 | 37981 | 31.6 |
| 96K- | ha120.dbf | 120190 | 34707 | 28.9 | 35689 | 29.7 | 38057 | 31.7 |
| | 120k1.dbf | 120802 | 16471 | 13.6 | 16787 | 13.9 | 17474 | 14.5 |
| | 120k2.dbf | 120802 | 16417 | 13.6 | 16695 | 13.8 | 17434 | 14.4 |
| | Avg | 120496 | 25572 | 21.2 | 26197 | 21.7 | 27737 | 23.0 |
| 190K | h190.dbf | 190156 | 54549 | 28.7 | 56042 | 29.5 | 59557 | 31.3 |
| 152K- | 190k2.dbf | 191170 | 25518 | 13.3 | 26123 | 13.7 | 27274 | 14.3 |
| 228K | 190k1.dbf | 191170 | 25516 | 13.3 | 26111 | 13.7 | 27087 | 14.2 |
| | Avg | 190832 | 35194 | 18.4 | 36092 | 18.9 | 37973 | 19.9 |
| 300K | 300k2.dbf | 301762 | 39899 | 13.2 | 40939 | 13.6 | 42518 | 14.1 |
| 240K- | 300k1.dbf | 301762 | 39984 | 13.3 | 40923 | 13.6 | 42462 | 14.1 |
| 360K | Avg | 301762 | 39942 | 13.2 | 40931 | 13.6 | 42490 | 14.1 |
| 500K | 500k2.dbf | 502818 | 65854 | 13.1 | 67719 | 13.5 | 70223 | 14.0 |
| 400K- | 500k1.dbf | 502818 | 66134 | 13.2 | 67808 | 13.5 | 70211 | 14.0 |
| 600K | Avg | 502818 | 65994 | 13.1 | 67764 | 13.5 | 70217 | 14.0 |
| 800K | zipco.dbf | 967384 | 258528 | 26.8 | 260192 | 27.0 | 290739 | 30.1 |
| 640K- | 800k2.dbf | 804450 | 104979 | 13.0 | 107826 | 13.4 | 111587 | 13.9 |
| 960K | 800k1.dbf | 804450 | 105141 | 13.1 | 107873 | 13.4 | 111703 | 13.9 |
| | Avg | 858761 | 156216 | 18.2 | 158630 | 18.5 | 171343 | 20.0 |

** For various sizes of dBASE Output Files, Compressed and Archived

| File Size | dBASE | ARJ221A | LHA213 | PK251 |
|---|---|---|---|---|
| Total | 5,937,002 | 1,051,036 | 1,069,782 | 1,144,139 |
| Ratio | 100 % | 17.7 % | 18.0 % | 19.3 % |

Table 8 COMPRESSED AND ARCHIVED, IMAGE FILES < Fig.12 >

** For various sizes of Image(Graphic) Files, Compressed and Archived

| File Size | | Image | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|---|
| 0.5K | augus.svg | 494 | 111 | 22.5 | 111 | 22.5 | 115 | 23.3 |
| 400- | bushh.svg | 494 | 107 | 21.7 | 107 | 21.7 | 107 | 21.7 |
| 600 | pebbl.svg | 494 | 122 | 24.7 | 122 | 24.7 | 131 | 26.5 |
| | grchk.f | 599 | 326 | 54.4 | 326 | 54.4 | 380 | 63.4 |
| | compa | 460 | 219 | 47.6 | 219 | 47.6 | 295 | 64.1 |
| | Avg | 508 | 177 | 34.8 | 177 | 34.8 | 206 | 40.6 |
| 1K | free1.wpg | 1210 | 612 | 50.6 | 612 | 50.6 | 767 | 63.4 |
| 800- | mktbl | 1044 | 615 | 58.9 | 615 | 58.9 | 774 | 74.1 |
| 1200 | grlgt.f | 877 | 455 | 51.9 | 455 | 51.9 | 556 | 63.4 |
| | pgcp.f | 893 | 433 | 48.5 | 433 | 48.5 | 535 | 59.9 |
| | pglab.f | 924 | 397 | 43.0 | 397 | 43.0 | 534 | 57.8 |
| | Avg | 990 | 502 | 50.7 | 502 | 50.7 | 633 | 63.9 |
| 1.8K | free3.wpg | 1422 | 672 | 47.3 | 672 | 47.3 | 872 | 61.3 |
| 1440- | fhvst.wpg | 1916 | 1030 | 53.8 | 1030 | 53.8 | 1419 | 74.1 |
| 2160 | free5.wpg | 1644 | 689 | 41.9 | 689 | 41.9 | 1000 | 60.8 |
| | free6.wpg | 1618 | 741 | 45.8 | 741 | 45.8 | 1040 | 64.3 |
| | Avg | 1650 | 783 | 47.5 | 783 | 47.5 | 1083 | 65.6 |
| 3K | snow.rf | 3478 | 1269 | 36.5 | 1269 | 36.5 | 1527 | 43.9 |
| 2400- | patti.shp | 2432 | 676 | 27.8 | 677 | 27.8 | 969 | 39.8 |
| 3600 | headc. | 2842 | 1304 | 45.9 | 1304 | 45.9 | 1535 | 54.0 |
| | metal | 2536 | 1324 | 52.2 | 1324 | 52.2 | 1483 | 58.5 |
| | fjamm.wpg | 2412 | 1142 | 47.3 | 1142 | 47.3 | 1560 | 64.7 |
| | grap2.wpg | 3558 | 1126 | 31.6 | 1124 | 31.6 | 1541 | 43.3 |
| | Avg | 2876 | 1140 | 39.6 | 1140 | 39.6 | 1436 | 49.9 |
| 5K | verti.vrs | 4945 | 1363 | 27.6 | 1363 | 27.6 | 1828 | 37.0 |
| 4K- | fonti.shp | 4096 | 1630 | 39.8 | 1627 | 39.7 | 1937 | 47.3 |
| 6K | haa1.dwg | 4368 | 1518 | 34.8 | 1505 | 34.5 | 1959 | 44.8 |
| | colo | 5860 | 2435 | 41.6 | 2436 | 41.6 | 2678 | 45.7 |
| | garfi.im1 | 4961 | 2275 | 45.9 | 2275 | 45.9 | 2575 | 51.9 |
| | grope.f | 4538 | 1666 | 36.7 | 1666 | 36.7 | 1913 | 42.2 |
| | Avg | 4795 | 1815 | 37.9 | 1812 | 37.8 | 2148 | 44.8 |
| 8K | e3830.dwg | 8464 | 2336 | 27.6 | 2336 | 27.6 | 2897 | 34.2 |
| 6400- | etbl | 8379 | 3199 | 38.2 | 3201 | 38.2 | 3516 | 42.0 |
| 9600 | imdri.f | 8942 | 2682 | 30.0 | 2676 | 29.9 | 3058 | 34.2 |
| | sunvi.sha | 7685 | 2830 | 36.8 | 2832 | 36.9 | 3128 | 40.7 |
| | syn.me | 9147 | 2473 | 27.0 | 2463 | 26.9 | 2764 | 30.2 |
| | teapo | 8227 | 2705 | 32.9 | 2686 | 32.6 | 3079 | 37.4 |
| | Avg | 8474 | 2704 | 31.9 | 2699 | 31.9 | 3074 | 36.3 |

** For various sizes of Image(Graphic) Files, Compressed and Archived

| File Size | Image | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|
| 13K | arch.dat | 13717 | 2577 | 18.8 | 2580 | 18.8 | 3261 | 23.8 |
| 10.4- | bearl.rf | 15200 | 3410 | 22.4 | 3342 | 22.0 | 4061 | 26.7 |
| 15.6 | geniu.vrs | 12361 | 3528 | 28.5 | 3533 | 28.6 | 4309 | 34.9 |
| | main.shp | 11264 | 4694 | 41.7 | 4684 | 41.6 | 5580 | 49.5 |
| | thesi.dwg | 11984 | 4321 | 36.1 | 4293 | 35.8 | 4919 | 41.0 |
| | Avg | 12905 | 3706 | 28.7 | 3686 | 28.6 | 4426 | 34.3 |
| 20K | clown.rf | 17816 | 5541 | 31.1 | 5450 | 30.6 | 6379 | 35.8 |
| 16K- | turk.rf | 19288 | 8756 | 45.6 | 8659 | 44.9 | 9772 | 50.7 |
| 24K | aero.eps | 21577 | 6209 | 28.8 | 6243 | 28.9 | 7021 | 32.5 |
| | bord.shp | 20608 | 7250 | 35.2 | 7185 | 34.9 | 8231 | 39.9 |
| | tsai.dwg | 18688 | 7625 | 40.8 | 7584 | 40.6 | 8630 | 46.2 |
| | Avg | 19595 | 7074 | 36.1 | 7024 | 35.8 | 8007 | 40.9 |
| 40K | golf.dat | 50186 | 6366 | 12.7 | 6218 | 12.4 | 8168 | 16.3 |
| 32K- | birds.rf | 47865 | 16217 | 33.9 | 15919 | 33.3 | 17627 | 36.8 |
| 48K | scree.rf | 38147 | 2263 | 5.9 | 2205 | 5.8 | 3274 | 8.6 |
| | sql.sha | 44976 | 11727 | 26.1 | 11888 | 26.4 | 12656 | 28.1 |
| | img8.rgb | 30752 | 3872 | 12.6 | 3870 | 12.6 | 5275 | 17.2 |
| | Avg | 42385 | 8089 | 19.1 | 8020 | 18.9 | 9400 | 22.2 |
| 70K | slib.shp | 70400 | 39370 | 55.9 | 38899 | 55.3 | 41435 | 58.9 |
| 56K- | bv.sr | 84432 | 63456 | 75.2 | 63601 | 75.3 | 64493 | 76.4 |
| 84K | bfg.sr | 77089 | 59117 | 76.7 | 59088 | 76.6 | 60110 | 78.0 |
| | show | 82177 | 29309 | 35.7 | 29616 | 36.0 | 31853 | 38.8 |
| | xhip | 82174 | 29839 | 36.6 | 30166 | 36.7 | 32408 | 39.4 |
| | Avg | 79254 | 44218 | 55.8 | 44274 | 55.9 | 46060 | 58.1 |
| 120K | augus.m18 | 111864 | 24492 | 21.9 | 24171 | 21.6 | 29989 | 26.8 |
| 96K- | bush.m18 | 111864 | 22129 | 19.8 | 21832 | 19.5 | 27710 | 24.8 |
| 144K | peb.m18 | 111864 | 22025 | 19.7 | 21701 | 19.5 | 27546 | 24.6 |
| | lenno.im1 | 129632 | 32916 | 25.4 | 31792 | 24.5 | 35356 | 27.3 |
| | movie | 98563 | 36348 | 36.9 | 36781 | 37.3 | 39400 | 40.0 |
| | space.im1 | 129700 | 20063 | 15.5 | 19162 | 14.8 | 23144 | 17.8 |
| | Avg | 115581 | 26329 | 22.8 | 25907 | 22.4 | 30524 | 26.4 |
| 190K | plant.mif | 177980 | 25647 | 14.4 | 25365 | 14.3 | 29750 | 16.7 |
| 152K- | bdy2.cbd | 228799 | 13482 | 5.9 | 109041 | 47.7 | 112064 | 49.0 |
| 228K | img5.rgb | 223800 | 166283 | 74.3 | 169196 | 75.6 | 174185 | 77.8 |
| | img9.rgb | 200427 | 121321 | 60.5 | 121977 | 60.9 | 124726 | 62.2 |
| | img14.eps | 176370 | 67453 | 38.2 | 67509 | 38.3 | 70915 | 40.2 |
| | Avg | 201475 | 78837 | 39.1 | 98618 | 48.9 | 102184 | 50.7 |
| 300K | ad.eps | 320174 | 98293 | 30.7 | 98369 | 30.8 | 106920 | 33.4 |
| 240K- | img13.rle | 243696 | 128686 | 52.8 | 130106 | 53.4 | 133342 | 54.1 |
| 360K | bdy.cbd | 261981 | 128032 | 48.9 | 128538 | 49.1 | 133263 | 50.9 |

** For various sizes of Image(Graphic) Files, Compressed and Archived

| File Size | Image | ARJ221A | | LHA213 | | PAK251 | |
|---|---|---|---|---|---|---|---|
| libpg.a | 362473 | 131703 | 36.3 | 133766 | 36.9 | 146741 | 40.5 |
| Avg | 297081 | 121679 | 41.0 | 122695 | 41.3 | 130067 | 43.8 |
| 500K 944gt.scr | 494267 | 192322 | 38.9 | 194529 | 39.4 | 207753 | 42.0 |
| 400K-bab02.eps | 526772 | 157291 | 29.9 | 156857 | 29.8 | 166791 | 31.7 |
| 600K 63vet.scr | 471937 | 159396 | 33.8 | 162441 | 34.4 | 171892 | 36.4 |
| b2.scr | 424532 | 269599 | 63.5 | 274615 | 64.7 | 283949 | 66.9 |
| Avg | 479377 | 194652 | 40.6 | 197111 | 41.1 | 207596 | 43.3 |
| 800K bigk.scr | 767399 | 225720 | 29.4 | 234851 | 30.6 | 249155 | 32.5 |
| 640K- ball.scr | 742684 | 355815 | 47.9 | 364433 | 49.1 | 379561 | 51.1 |
| 960K beac.scr | 803894 | 485758 | 60.4 | 493360 | 61.4 | 562386 | 69.9 |
| half.scr | 961208 | 591612 | 61.5 | 604234 | 62.9 | 658037 | 68.5 |
| solin.sc | 1070111 | 754167 | 70.5 | 775119 | 72.4 | 825469 | 77.1 |
| Avg | 869059 | 482614 | 55.5 | 494399 | 56.9 | 534922 | 61.6 |
| Total | 10,033,651 | 4,586,482 | | 4,758,203 | | 5,207,268 | |
| Ratio | 100 % | 45.7 % | | 47.4 % | | 51.9 % | |

## APPENDIX C. EXECUTION TIME COMPARISON

| Sample Files | | PKZIP | ARJ221A | LHA213 | PAK251 |
|---|---|---|---|---|---|
| inst.doc | 4029 | 123456 | 123456 | 123456 | 123456 |
| readm.txt | 5594 | | | | |
| grep2.exe | 5934 | | | | |
| touch.com | 5118 | | | | |
| sysco.dbf | 5586 | | | | |
| dba4.dbf | 4969 | | | | |
| verti.vrs | 4945 | | | | |
| haa1.dwg | 4368 | | | | |
| Total | **40543** | 1.5 | 3.8 | 3.0 | 4.2 |
| api.doc | 15240 | | | | |
| read3.doc | 12006 | | | | |
| mips.com | 13312 | | | | |
| share.exe | 13424 | | | | |
| dba1.dbf | 12639 | | | | |
| offic.dbf | 11261 | | | | |
| arch.data | 13717 | | | | |
| Total | **91599** | 3.1 | 4.7 | 3.6 | 4.4 |
| chara.doc | 42223 | | | | |
| parts.hlp | 33583 | | | | |
| dcm.exe | 45212 | | | | |
| copy.exe | 42398 | | | | |
| h40k.dbf | 40240 | | | | |
| 40k_1.dbf | 40354 | | | | |
| birds.rf | 47865 | | | | |
| img8.rgb | 30752 | | | | |
| Total | **322627** | 10.0 | 10.7 | 8.6 | 10.5 |
| tex.lib | 131653 | | | | |
| lin.lib | 110682 | | | | |
| ll3.exe | 93399 | | | | |
| ift.exe | 111894 | | | | |
| h120.dbf | 120190 | | | | |
| 120k1.dbf | 120802 | | | | |
| augus.m18 | 111864 | | | | |
| movie | 98563 | | | | |
| Total | **899047** | 19.3 | 21.5 | 18.9 | 21.5 |
| quatt.hlp | 287589 | | | | |
| mcad.exe | 289664 | | | | |
| check.exe | 351232 | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 300k2.dbf | 301762 | | | | |
| 300k1.dbf | 301762 | | | | |
| ad.eps | 320174 | | | | |
| img13.rle | 243696 | | | | |
| Total | **2095879** | 44.1 | 1:01.6 | 52.0 | 1:01.5 |

| | | | | | |
|---|---|---|---|---|---|
| tchel.tch | 976250 | | | | |
| pshel.exe | 635552 | | | | |
| tc.exe | 887104 | | | | |
| 800k2.dbf | 804450 | | | | |
| 800k1.dbf | 804450 | | | | |
| half.scr | 961208 | | | | |
| solin.sc | 1070111 | | | | |
| Total | **6139125** | 1:57.4 | 3:20.9 | 2:41.7 | 2:33.2 |

## LIST OF REFERENCES


1. V. Cappellini, "Data Compression and Error Control Techniques With Applications," Academic Press, 1985.

2. T. C. Bell, "Better OPM/L Text Compression," IEEE Trans. Commun., vol. COM-34, no. 12, PP. 1176-1182, Dec. 1986.

3. B. Simon, "Squeeze Play(Software Review)," PC Magazine, vol. 10, no. 17, PP. 291-300, PP. 316-317, Oct. 1991.

4. T. A. Welch, "A Technique for High Performance Data Compression," IEEE Computer, vol. 17, no. 6, PP. 8-19, Jun. 1984.

5. P. E. Bender and J. K. Wolf, "New Asymptotic Bounds and Improvements on the Lempel-Ziv Data Compression," IEEE Trans. Inform. Theory, vol. 37, no. 3, PP. 721-729, May 1991.

6. "compress(1)," UNIX Programmer's Man., 4.3 Berkeley Distribution, May 1986.

7. R. A. Monsour and D. L. Whiting, "Data Compression Breaks Through to Disk Memory Technology," Computer Technology Review , PP. 39-44, Spring 1991.

8. "PKZIP, appnote.txt,"PKWARE Inc., Aug. 1991.

9. G. Held and T. R. Marshall, "Data Compression, Techniques, and Applications, Hardware and Software Considerations," John Wiley & Sons, 1987.

10. D. A. Lelewer and D. S. Hirschberg, "Data Compression," ACM Computing Surveys, vol. 19, no. 3, PP. 262-296, Sept. 1987.

11. K. Anderson, "Overview of Huffman Encoding as Compression Technique," Computer Tech. Review, PP. 97-101, Spring 1991.

12. S. J. Vaughan-Nichols, "Data Compression: Making Space For Today's Applications," Personal Workstation, vol. 3, no. 4, PP. 50-55, April 1991.

13. D. Wiseman and B. Miller, "The Technology of Data Compression and Its Benefits to HP3000 Users,"

SuperGroup Magazine, vol. 11, no. 3, PP. 16-19, May-June 1991.

14. R. K. Jung, "User's Manual For the ARJ Archiver Programs," BBS, Oct. 1991.

15. "PAK251, PAK.doc", NoGate Consulting, BBS, 1990.

16. H. Yoshizaki, "Manual For LHA Version 2.13," BBS, Jul. 1991.

17. M. Dufort, "Getting The Least Out Of Your Data," Computer Language, vol. 8, no. 5, PP. 45-57, Dec. 1991.

18. J. A. Storer and T. G. Szymanski, "Data Compression Via Textual Substitution," J. ACM, vol. 29, no. 4, PP. 928-951, 1982.

19. Y. Perl, V. Maram, and N. Kadakuntla, "The Cascading Of The LZW Compression Algorithm With Arithmetic Coding,"IEEE Computer Society Press, PP. 277-286, Data Compression Conference, Snowbird, Utah, April 8-11, 1991.

20. H. Yokoo, "An Improvement of Dynamic Huffman Coding With a Simple Repetition Finder," IEEE Trans. Commun., vol. 39, no.1, PP. 8-10, Jan. 1991.

21. M. R. Nelson, "Arithmetic Coding and Statistical Modeling: Achieving Higher Compression Rates," Dr.Dobb's Journal, vol. 16, no. 2, PP. 16-27, Feb. 1991.

22. "ARC, File Archive Utility Version 6.00," System Enhancement Associates, Inc., BBS, Jan. 1989.

23. R. Dhesi, "ZOO(1) Reference Manual," BBS.

24. "Data Compression Research Requirements," The Naval Security Group Detachment, Pensacola, Florida, Apr. 1991.

25. "MathCad, Ver. 2.50," MathSoft, Jun. 1989.

26. "386 Matlab, Ver. 3.5j," MathWorks, May 1991.

27. "PSpice, Ver. 4.05," MicroSim, Feb. 1991.

28. "WordPerfect, Ver. 5.1," WordPerfect Corp., Mar. 1990.

29. "DASH-4, Ver. 4.02m," FutureNet, Mar. 1988.

30. "MS-DOS, Ver. 4.01," MicroSoft, 1988.

31. "Turbo C++, Ver. 2.0," Borland, Oct. 1991.

32. "dBASE IV," Ashton-Tate, 1988.

33. "DrawPerfect, Ver. 1.0," WordPerfect Corp., Jun. 1990.

## INITIAL DISTRIBUTION LIST

| | | |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Library Code 52<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | 2 |
| 3. | Department Chairman, Code EC<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | 1 |
| 4. | Naval Security Group Detachment<br>Corry Station<br>Pensacola, FL 32511-5100 | 1 |
| 5. | Professor Chyan Yang, Code EC/Cw<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | 1 |
| 6. | Professor Glen Myers, Code EC/Mv<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | 1 |
| 7. | Jung, Young Je<br>155-17, 12-Tong 3-Ban, Jangjun 1-Dong,<br>Kumjung-Ku, Pusan,<br>Republic of Korea 609-391 | 1 |
| 8. | Ken Oh<br>Pacific Missile Test Center, Code 1052<br>Point Mugu, CA 93042 | 1 |
| 9. | Captain Doo Jong Kim<br>SMC 1587 NPGS<br>Monterey, CA 93943 | 1 |
| 10. | Chun Taek Lim<br>35 Jamshil-Dong, Apt. 367-Dong 202-Ho,<br>Songpa-Gu, Seoul,<br>Republic of Korea | 1 |

11. Library 1
Kum-Oh Institute of Technology,
180-1, Sinpyeong 1-Dong, Kumi-City, KyungPook-Do,
Republic of Korea 730-770

12. Library 1
P.O.Box 77, Gongneung-Dong, Dobong-Gu, Seoul,
Republic of Korea 132-240

13. Army Central Library 1
Army Headquarter, Bunam-Ri, Duma-Myun,
Nonsan-Gun, ChungNam-Do
Republic of Korea 320-919